BCPY: Toolkit para experimentação de sistemas Interface Cérebro-Computador

Igor Neves Faustino - 42885

Dissertação apresentada à Escola Superior de Tecnologia e de Gestão de Bragança para obtenção do Grau de Mestre em Sistemas de Informação no âmbito da dupla diplomação com a Universidade Tecnológica Federal do Paraná.

Trabalho orientado por:

Prof. Doutor Rui Pedro Lopes

Prof. Doutor Rodrigo Hübner

Bragança

Junho de 2020

BCPY: Toolkit para experimentação de sistemas Interface Cérebro-Computador

Igor Neves Faustino - 42885

Dissertação apresentada à Escola Superior de Tecnologia e de Gestão de Bragança para obtenção do Grau de Mestre em Sistemas de Informação no âmbito da dupla diplomação com a Universidade Tecnológica Federal do Paraná.

Trabalho orientado por:

Prof. Doutor Rui Pedro Lopes

Prof. Doutor Rodrigo Hübner

Bragança

Junho de 2020

Resumo

A interação do cérebro humano com sistemas de computadores se tornou possível devido ao avanço do conhecimento biológico e tecnológico atual. Experimentos que exploram esse conceito são chamados de Interface Cérebro-Computador (BCI). Esta dissertação tem como objetivo auxiliar o desenvolvimento de experimentos BCI em suas etapas fundamentais, de forma que facilite pesquisadores a trabalharem nesta área. Para tal, está sendo desenvolvido um *toolkit* para unificar as ferramentas necessárias para realizar os experimentos, permitindo que o pesquisador possa utilizar apenas a solução proposta. Além das básicas do experimento, também foi desenvolvido uma interface que permita acompanhar visualmente em tempo real a execução dos experimentos. Após os testes realizados com o toolkit desenvolvido, foi comprovado a capacidade da solução de auxiliar e simplificar a criação de experimentos BCI, sendo assim esperado que a solução incentive que novas pesquisas possam ser realizadas.

Palavras-chave: Interface Cérebro-computador, Toolkit, EEG.

Abstract

Using the human brain to interact with a computer system has become possible, this is possible due to the advancement of biological science and current technological updates. This research area is called the Brain Computer Interface (BCI). This thesis aims to help develop new BCI experiments in a way that facilitates researchers to start a new experiment. To achieve this goal, a toolkit is being developed to unify all the tools necessary to develop a BCI experiment, allowing the researcher to do all the work using only the solution. A GUI is also being developed, providing complete monitoring of the state of the experiment. After the tests carried out, the capacity of the solution to help and simplify new BCI experiments was verified. We hope that the tool can increase the number of experiments in this area.

Keywords: BCI, Toolkit, EEG.

Conteúdo

Lista de Tabelas

Lista de Figuras

Siglas

BCI *Brain-Computer Interface.* 1, 2, 5–9, 12–16, 19, 20, 23, 28, 29, 35, 43, 44, 49, 53

CAR *Common Average Reference.* 8

ECoG Eletrocorticografia. 6

EEG Eletroencefalografia. 6, 13–16, 19, 27, 28, 30, 33, 35

LSL *Lab Steaming Layer.* 6, 7, 31, 32, 44

MEG Magnetoencefalografia. 15, 16

PSD *Power Spectral Density.* 10

SSVEP *Steady State Evoked Potential.* 9, 10, 46–49

SVM *Support-Vector Machines.* 47

UML *Unified Modeling Language.* 24

Capítulo 1

Introdução

Nos últimos anos, o conhecimento das atividades cerebrais vem se tornando cada vez mais presente. Equipamentos que permitem a mensuração das atividades cerebrais já podem ser utilizados para identificar padrões e comportamentos dos seres humanos. Com o avanço da tecnologia tem-se tornado possível realizar interações entre o cérebro e sistemas de computadores de forma direta, sendo denominado Interface Cérebro-Computador (do inglês, *Brain-Computer Interface* (BCI)) [1]. Sistemas BCI consistem na utilização de sinais produzidos em diferentes regiões do cérebro, como fonte de comandos capazes de controlar ações dentro de uma aplicação [2].

Com o aumento de pesquisas BCI, uma maior quantidade de estudos estão gerando aplicações de caráter inclusivo, visando melhorar a qualidade de vida de pessoas com algum tipo de deficiência física, possibilitando uma nova interface para interagir com tecnologias já existentes [2].

Para a realização de novos experimentos BCI, diferentes conceitos e estratégias já estão definidos, ganhando o nome na literatura de paradigmas. Os paradigmas são responsáveis por definir qual será a abordagem utilizada em cada experimento, definindo quais serão os dados úteis extraídos e como serão processados e classificados.

Devido ao crescente interesse em experimentos utilizando tecnologias BCI, diferentes ferramentas estão sendo desenvolvidas para auxiliar pesquisadores em seus trabalhos.

No entanto, muitas destas ferramentas foram desenvolvidas com foco em problemas específicos, capazes de realizar de forma satisfatória pequenas etapas de um experimento. Assim, acabam necessitando de auxílio de outras tecnologias para compor o experimento completo. A grande quantidade de ferramentas, desenvolvidas em diferentes linguagens, com diferentes padrões resulta em uma complexidade extra para os pesquisadores. Outro ponto importante é a dificuldade em encontrar opções para o pesquisador acompanhar o sinal mensurado do cérebro em tempo real, sendo esta uma funcionalidade importante para verificar o estado do experimento e o funcionamento de todos os componentes.

1.1 Objetivo

Esta dissertação possui como o objetivo principal o desenvolvimento de um *toolkit* multiplataforma de código aberto, que possibilite pesquisadores realizarem todas as etapas de um experimento BCI, sem a necessidade de recorrer a ferramentas externas para realizar funções pertinentes ao experimento.

Foi desenvolvido uma ferramenta que contemple todas as etapas presentes em experimentos BCI, sendo estas: aquisição do sinal; pré-processamento; extração e seleção de características; e por fim a etapa de classificação. Foi focado durante a projeção deste *toolkit* uma solução que permita a visualização dos dados em tempo real durante a realização do experimento, uma função pouco presente em outras ferramentas existentes. O *toolkit* foi desenvolvido utilizando a linguagem de programação `Python` devido sua grande popularidade para resolver problemas científicos [3], enquanto o módulo de visualização foi desenvolvido utilizando a linguagem `javascript` por permitir uma maior compatibilidade entre diferentes sistemas operacionais.

1.2 Estrutura do Documento

Este trabalho está dividido em 7 capítulos. Além deste primeiro que apresenta uma introdução aos tópicos abordados, os demais capítulos têm a seguinte estrutura: no Capítulo 2

são apresentados os conceitos dos itens estudados para o desenvolvimento desta dissertação; o Capítulo 3 apresenta os trabalhos relacionados utilizados como base; no Capítulo 4 é apresentado a metodologia tomada para o desenvolvimento; o Capítulo 5 apresenta de forma detalhada o desenvolvimento realizado neste trabalho; No Capítulo 6 são apresentados os testes e experimentos realizados utilizando o *toolkit* desenvolvido; e por fim o Capítulo 7 apresenta as conclusões e trabalhos futuros.

Capítulo 2

Conceitos

Sistemas BCI apresentam de forma geral um esquema genérico para a maior parte das aplicações. Tal esquema pode ser definido pelos seguintes passos: aquisição do sinal; pré-processamento; extração e seleção de características; classificação; e por fim a aplicação informática (Figura 2.1).

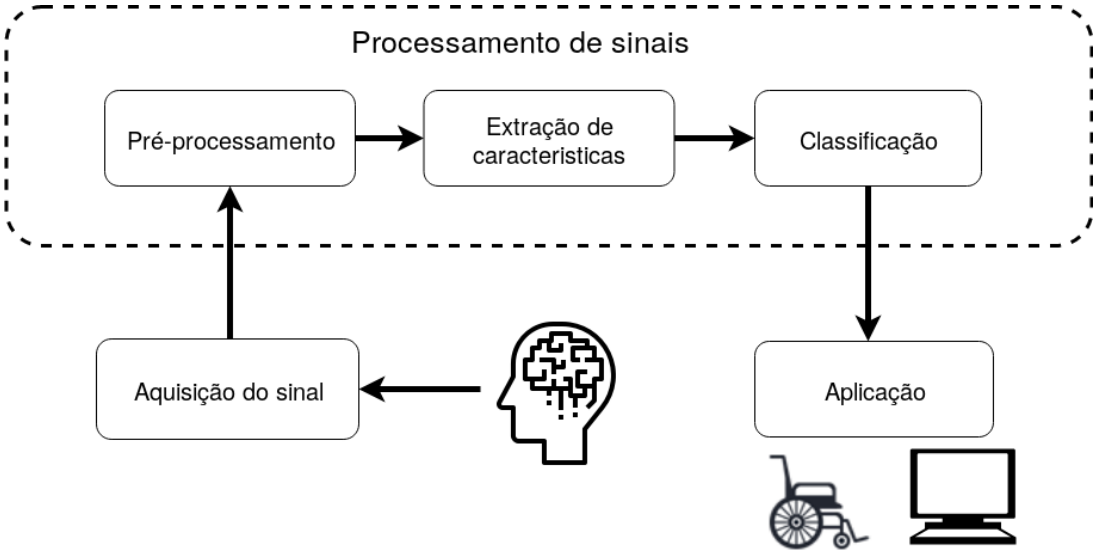

Figura 2.1: Esquema comum de um sistema BCI [4, p. 42].

Neste capítulo é apresentada a fundamentação teórica de cada etapa de um experimento BCI, tendo como foco demonstrar os fluxos e as soluções adotadas e utilizadas em cada etapa desta dissertação.

2.1 Aquisição do sinal

A aquisição de sinais BCI pode ser realizada por meio de diferentes métodos, sendo estes possíveis de serem invasivos ou não invasivos. Métodos invasivos necessitam a realização de uma cirurgia, com o intuído de posicionar os elétrodos diretamente na membrana que envolve o córtex cerebral, método este chamado de Eletrocorticografia (ECoG) [2]. Este método, em questão da qualidade do sinal, apresenta os melhores resultados, porém com o risco de gerar danos ao indivíduo [4].

Métodos não invasivos são considerados mais seguros e geralmente de baixo custo, todavia estes métodos obtêm sinais mais fracos devido a interferência dos tecidos que envolvem o cérebro e o crânio. De todas as formas não invasivas de obter sinais BCI, a mais utilizada é a Eletroencefalografia (EEG) devido a sua segurança e facilidade de uso [5]. A realização de um EEG consiste em posicionar elétrodos na cabeça do indivíduo a fim de mensurar a diferença de potencial gerado pelo conjunto de sinapses geradas no cérebro. A posição dos elétrodos deve ser escolhida de modo que todas as regiões do córtex cerebral relevantes para o experimento possam ser mensuradas. Um modelo padrão e bastante aceito para o posicionamento dos elétrodos é chamado de Sistema Internacional 10-20 [4], no qual os elétrodos são posicionados a uma distância de 10 ou 20% da região que vai de násion a ínion, como apresentado na Figura 2.2. Neste trabalho será focado em dados obtidos por EEG.

Após a aquisição do sinal por meio de hardwares específicos, a informação coletada necessita ser integrada de alguma forma com o sistema desenvolvido para o experimento. A integração pode ocorrer por meio de softwares proprietários disponibilizados pelo fabricante do hardware, os quais dificilmente são multiplataforma, ou utilizam soluções de código aberto para tal fim.

Neste trabalho, para realizar a comunicação com o EEG, foi utilizado principalmente o protocolo *Lab Steaming Layer* (LSL), para permitir a transmissão de dados em tempo real, tratando problemas de rede e de sincronização [7]. Este protocolo está disponibilizado em sua grande maioria pela licença MIT e foi adotado por diversos hardwares disponíveis

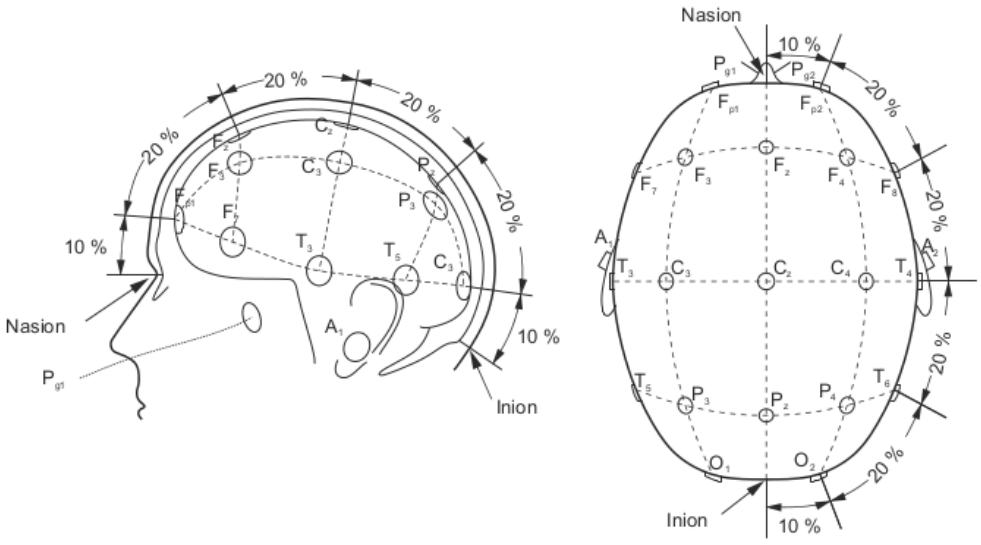

Figura 2.2: Mapeamento dos elétrodos sobre o escalpo seguindo o sistema 10-20 [6].

para experimentos BCI como uma de suas opções de transmissão, permitindo assim uma integração do *toolkit* proposto com diferentes tipos de hardwares. Além da utilização do protocolo LSL, o *toolkit* desenvolvido permite a utilização de métodos customizados pelo usuário, com o objetivo de permitir com que mais protocolos e dados salvos em diferentes tipos de arquivo sejam contemplados.

2.2 Pré-processamento

O processamento de sinais é uma etapa de suma importância para aplicações BCI, ajudando a isolar a informação útil dos dados cerebrais brutos obtidos dos sensores [8] e remover artefatos, que representam informações inesperadas que não possuem relação com a informação utilizada pelo experimento, sendo provido de fontes externas como: o piscar dos olhos; uma respiração não natural realizada pelo indivíduo; ranger dos dentes; movimentos musculares; etc [9]. Para realizar o pré-processamento dos dados são utilizados filtros temporais e espaciais, destinados a alterarem os dados de formas específicas para atenuar ou realçar certas informações desejadas.

Filtros temporais são utilizados para remover frequências desnecessárias para serem

utilizadas nas demais etapas do sistema BCI. Este processo pode ser realizado de maneira física ou posteriormente por software após os dados serem digitalizados [9]. Exemplos de filtro temporais incluem: passa-baixa, que atenuam altas frequências sem alterar as baixas frequências; passa-alta, que atenuam baixas frequências sem alterar as altas; passa-faixa, que atenua tudo o que não está dentro de uma determinada faixa de frequência; e os filtros *notch*, responsáveis por atenuarem uma determinada frequência específica.

Sistemas BCI sofrem de interferências constantes de fontes externas, se tornando altamente recomendado a aplicação de filtros temporais. Para exemplificar, frequências abaixo de 5 Hz não possuem sinais úteis enquanto frequências próximas de 50 Hz ou 60 Hz, dependendo do país, podem sofrer de interferência da rede elétrica local, gerando uma grande quantidade de artefatos nos dados. Desta forma, se torna comum a utilização de filtros passa-faixa entre os valores de 5 a 40 Hz [9].

Filtragens espaciais têm como o objetivo reduzir a dimensionalidade dos dados e realçar um conjunto específico de elétrodos. Em sistemas BCI, este filtro significa reduzir a quantidade de elétrodos analisado, ou mesmo remover dados em comuns entre diferentes canais. Dessa forma, são mantidos apenas os dados que diferem dos demais, sendo provavelmente os mais relevantes para a maioria das aplicações [10]. Durante a realização deste projeto, foi implementado os filtros temporais passa-baixa, passa-alta, passa-faixa e *notch*, utilizando um filtro `FIR` [11] e como filtro temporal foi implementado o filtro *Common Average Reference* (CAR) para remover dados comuns a todos os elétrodos [12].

2.2.1 Extração de características

O processo de traduzir os valores obtidos do cérebro de um indivíduo para informações úteis, que possam ser utilizadas como comando de aplicações, é chamado de extração de características [9]. A extração de características consiste em mapear os dados brutos recebidos para um novo conjunto que represente de forma eficiente as informações relevantes para o experimento. Entretanto a tarefa de extrair informações relevantes do sinal não é trivial, devendo tomar o cuidado de não descartar dados úteis para o experimento [1].

As características relevantes para cada tipo de experimento podem variar, sendo definidas pelo tipo de paradigma escolhido para o experimento. Os paradigmas são os responsáveis por determinar quais serão as características relevantes para um determinado experimento, geralmente utilizando do comportamento conhecido da fisionomia do cérebro humano para definir quais são os dados mais relevantes de cada experimento [1].

2.2.2 Paradigma de atenção seletiva

Para exemplificar um paradigma BCI, será apresentado o paradigma de atenção seletiva. Este paradigma parte do conhecimento que estímulos externos conseguem evocar diferentes reações pelo cérebro, sendo possível mensurar essas reações e utilizar em experimentos [2]. Os estímulos podem ser visuais, auditivos ou táteis. Segundo o paradigma de atenção seletiva, o indivíduo deve focar (de acordo com o estímulo proposto pelo experimento) em um determinado estímulo alvo, o qual pode ser identificado ao analisar as reações cerebrais durante o momento do foco. Cada estímulo pode ser rotulado com um comando distinto, permitindo que o indivíduo possa selecionar o comando desejado apenas ao focar no estímulo correspondente.

Neste trabalho foi utilizado os estímulos visuais para a realização do experimento. Os principais paradigmas presentes neste segmento são: Potenciais Evocados Visualmente em Regime Estacionário (do inglês *Steady State Evoked Potential* (SSVEP)) e P300.

SSVEP

O paradigma SSVEP parte do conhecimento que estímulos visuais causam reações nas atividades cerebrais dos seres humanos, sendo possível a identificação de padrões cerebrais associados a cada estímulo e suas propriedades, como frequência e contraste [1]. Os neurónios no córtex visual respondem aos estímulos sincronizando suas atividades com a frequência relacionada ao estímulo visual, logo estímulos visuais de pelo menos 6 Hz desencadeiam uma resposta nas atividades cerebrais, aumentando a atividade cerebral evocada na mesma frequência no córtex visual. Desta forma, caso um indivíduo esteja

olhando para uma determinada frequência, esta pode ser detectada ao transformar o sinal cerebral obtido para o domínio da frequência por meio de uma transformada de Fourier [13], pois nesse paradigma a informação é concentrada em maior forma no espectro do sinal [1].

Um paradigma SSVEP é um sistema BCI que utiliza das propriedades discutidas para produzir padrões conhecidos que possam ser interpretados por um conjunto de programas, consistindo em um conjunto de estímulos com frequências diferentes, associadas a determinados comandos a serem executados pela aplicação. Como forma de evocar os estímulos visuais, é possível utilizar tanto LEDs como a própria tela de um computador. Enquanto os LEDs podem estimular qualquer frequência, grande parte dos sistemas utilizam as telas pela possibilidade de centralizar o conjunto de estímulos e o *feedback* do sistema em um único lugar. Porém, este meio está limitado a geração de frequências relacionadas a taxa de atualização da própria tela [13]. O fluxo básico de um experimento SSVEP é apresentado na Figura 2.3.

Classicamente, em experimentos SSVEP, os dados utilizados são extraídos da densidade espectral de potencia (do inglês *Power Spectral Density* (PSD)) próximos da frequência evocada. O PSD de maneira simples consiste nos valores absolutos obtidos por uma transformada de Fourier e sua utilização é justificada pela praticidade de identificar picos e informações relevantes para o experimento [1]. Diversos métodos podem ser utilizados para obter o PSD de um sinal, nesta tese foi utilizado o método de Welch [15].

2.2.3 P300

P300 é um paradigma que pode ser visualizado nos dados no domínio do tempo, devido a uma reação a um sinal visual externo classificada como um pico de onda positivo após um curto espaço de tempo (de 300 a 500ms) após um estímulo visual [4].

Utilizando o paradigma P300, é possível observar quando um sinal foi observado pelo indivíduo, e relacionar o período em que foi visualizado o estímulo com um determinado comando no sistema.

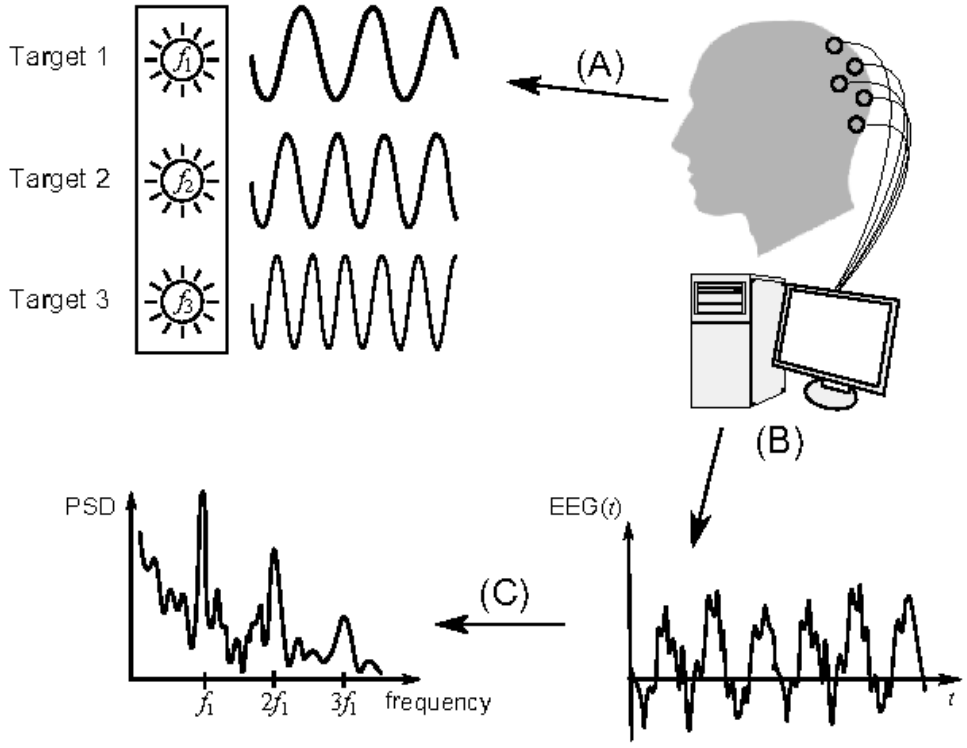

Figura 2.3: Exemplo de um experimento SSVEP [14]. (A) indivíduo com foco em f1; (B) Aquisição do sinal no domínio do tempo; (C) Transformação do sinal para o domínio da frequência demonstrando picos em f1, 2f1 e 3f1, indicando que o indivíduo está focando no alvo 1.

2.3 Classificação

A etapa de classificação dos dados consiste na utilização de classificadores e outros algoritmos de aprendizagem de máquina com o intuito de identificar a intenção do indivíduo e traduzi-la para um comando válido dentro do sistema [1]. Para realizar este processo são, normalmente, utilizadas as características extraídas nas etapas anteriores. Estas características devem separar cada dado em uma classe diferente, sendo que comumente neste escopo de pesquisa os dados pertencem a apenas uma única classe, os tornando disjuntos.

Uma característica limitante para esta linha de pesquisa é a quantidade dos dados. Como os dados obtidos são referentes a funções fisiológicas, que ocorrem de formas diferentes em cada pessoa, existe assim, uma dificuldade em utilizar dados gerais para treinar

um classificador, limitando os dados úteis para cada experimento aos dados coletados apenas do atual indivíduo que está participando.

Dificilmente, devido a quantidade dos dados, algoritmos de aprendizagem profunda podem ser aplicados, sendo a principal escolha para experimento BCI a utilização de classificadores lineares como o `SVM` [16] e o `LDA` [17].

2.4 Aplicações BCI

Por fim, a última etapa de um sistema BCI é o *feedback* passado para o usuário, dependendo do tipo de aplicação que está sendo desenvolvido. Diferentes aplicações podem ser encontradas utilizando paradigmas BCI, variando de aplicações visando a acessibilidade de indivíduos na sociedade a jogos eletrônicos. Dentre as aplicações BCI, podem ser citadas cadeiras de rodas automáticas, em que permite um usuário com deficiência motora controlar o funcionamento de uma cadeira de rodas por meio de paradigmas BCI [18], teclados virtuais, que permitem a digitação de textos por meio do foco do individuo e do paradigma P300 [19] e controle de diversos aparelhos eletrônicos em geral [20].

2.5 Considerações finais

Neste capítulo foi apresentado uma fundamentação dos assuntos trabalhos em experimentos BCI, além de introduzir conceitos importantes para o entendimento geral deste presente trabalho. Foi apresentado uma visão geral de todas as principais etapas de um experimento, as quais foram contempladas durante o desenvolvimento do *toolkit* proposto neste trabalho.

Capítulo 3

Trabalhos Relacionados

O interesse em pesquisas na área de BCI promove com que mais pesquisas e trabalhos sobre o desenvolvimento de ferramentas para auxiliar a experimentação de sistemas BCI surjam. Neste capítulo são descritas algumas das principais soluções disponíveis como software livre para a comunidade, estudados e tidos como inspiração e base para o desenvolvimento do *toolkit* proposto.

3.1 Design e Implementação de um sistema de Interface Cérebro-Computador

O trabalho titulado originalmente de "*Design and Implementation of a Brain-Computer Interface System*" [21] apresenta o desenvolvimento de uma biblioteca completa para a realização de sistemas BCI, na qual é apresentada uma estrutura e um fluxo completo bem definido, utilizando diversas máquinas conectadas por uma rede para delegar as diferentes etapas do experimento para cada hardware e outras ferramentas e protocolos para a realização dessa comunicação. É apresentado uma solução completa e fechada, que apesar de poder ser expandida para vários hardwares EEG, de forma geral apresenta uma única solução para a integração entre a aquisição de dados e a aquisição de marcadores provindos da geração de estímulos, sendo necessário a utilização do *framework* de criação

de aplicações de feedback chamado *pyff*.

Nesta tese, foi desenvolvido um conjunto de ferramentas BCI que visa permitir um fluxo de experimento simplificado, dependendo o mínimo possível de ferramentas externas que não possam ser integradas diretamente ao projeto. É visado suportar os principais paradigmas BCI, mas não limitados a eles, permitindo a expansão e integração de outras soluções não contempladas ao escopo atual.

3.2 Gumpy: ferramentas para sistemas BCI

No trabalho "*Gumpy: a Python toolbox suitable for hybrid brain–computer interfaces*" [22] é proposto um conjunto de ferramentas implementado em `Python` para possibilitar a realização de experimentos BCI. A ferramenta desenvolvida está contida em seis módulos referentes a plotagem dos dados, processamento e classificação, incluindo algoritmos de aprendizagem profunda e outros modelos. A ferramenta permite a realização de experimentos offline, em que não exite a necessidade de uma resposta imediata e experimentos em tempo real, na qual existe uma necessidade de repassar um *feedback* para o usuário de forma rápida dependendo da aplicação implementada.

É proposto então um trabalho no escopo do desenvolvimento de ferramentas para sistemas BCI, porém, apresentando carências que foram trabalhadas nesta dissertação. Sendo a principal delas a possibilidade de realizar a visualização dos dados obtidos pelo EEG em tempo real, ao mesmo tempo que permite o experimento ser realizado de forma online ou offline, permitindo assim ter uma maior noção sobre o experimento que está sendo realizado.

3.3 OpenBCI GUI

"*OpenBCI: Framework for Brain-Computer Interface*"[23] apresenta um software OpenBCI, desenvolvido para uma tese de mestrado. O software apresenta uma grande quantidade de funcionalidades, como por exemplo: aplicar amplificadores ao sinal obtido; visualizar

os dados em tempo real; e realizar o pré processamento dos dados.

O software OpenBCI foi uma solução desenvolvida utilizando a linguagem de programação `Java` e foi desenvolvido para cobrir todas as necessidades básicas de sistemas BCI. É utilizado para coordenar as informações obtidas pelo software, uma arquitetura centralizada, na qual todos os módulos comunicam com uma unidade central que encaminha toda a informação para seu respectivo destino.

Devido ao software OpenBCI ser desenvolvido utilizando a linguagem de programação `Java`, apresenta algumas limitações de integração com experimentos e bibliotecas já desenvolvidas, visto que, `Python` é atualmente uma das linguagens mais utilizadas para experimentos científicos e análise de dados atualmente [24]. Devido a simples sintaxe e facilidade de aprendizado da linguagem `Python` [25], pesquisadores que não estão familiarizados com linguagens de programação podem entender o funcionamento dos algoritmos desenvolvidos. Nesta dissertação foi desenvolvido uma solução desenvolvida principalmente em `Python` utilizando o padrão de desenvolvimento definido pela comunidade científica da linguagem, propiciando assim uma maior familiaridade para os pesquisadores da área.

3.4 MNE-Python

`MNE-Python` [26] é um subprojeto desenvolvido em cima do software acadêmico `MNE` [27] e tem como objetivo prover um conjunto de algoritmos destinados a encapsular grande parte de um processo de experimentos de Magnetoencefalografia (MEG) e EEG. Este projeto possui o intuito de manter sua usabilidade similar aos demais projetos disponibilizados pela comunidade `Python`, além de possibilitar uma interface entre o `MNE` original, devido a utilização do mesmo formato dos dados.

`MNE-Python` é um pacote baseado em *scripts* que permite a execução de diferentes formas de processamento e visualização do sinal trabalhado em diferentes pontos da aplicação. A abordagem em *scripts* foi realizada de maneira simples, permitindo assim os usuários utilizarem os recursos da `Python` para desenvolver os experimentos.

A partir de testes realizados de forma empírica no projeto `MNE-Python`, foi constatado o grande foco em experimentos utilizando MEG, que possui parâmetros diferentes de outros hardware, aumentando a complexidade ao realizar experimentos EEG, devido a grande quantidade de parâmetros que devem ser acertados para obter resultados satisfatórios. Nesta dissertação, foi desenvolvido um projeto com foco em experimento utilizando EEG, capaz de englobar os principais paradigmas da área e que possa, de forma simples, utilizando uma seleção de algoritmos já parametrizados com configurações necessárias para cada tipo de experimento proposto.

3.5 OpenVibe

`OpenVibe` [28] é um projeto de código livre destinado ao desenvolvimento de experimentos BCI, possuindo ferramentas modulares que permitem o reuso das mesmas, além de permitir a adição de novos módulos para suprir as necessidades dos usuários. `OpenVibe` foi desenvolvido utilizando a linguagem `C++`, porém, é disponibilizado uma interface gráfica onde este pode simplesmente ligar cada ação a ser realizada (representada por blocos) montando o fluxo que o experimento deve tomar para atingir sua completude, sendo seu maior diferencial a possibilidade de desenvolver um experimento sem a necessidade de escrever qualquer linha de código.

Apesar da existência de uma interface gráfica que é capaz de suprir os usuários menos experientes, caso exista a necessidade de expandir as funcionalidades disponíveis, é necessário entender toda a estrutura de classes desenvolvida em `C++`, dificultando a realização para pesquisadores menos experientes. No *toolkit* desenvolvido nesta tese, foi utilizado de uma linguagem mais simples para novos pesquisadores, além de manter uma estrutura básica preparada para poder ser expandida, utilizando padrões de projeto conhecidos.

3.6 Considerações finais

Nos trabalhos relacionados estudados, foi possível levantar uma grande quantidade de informações úteis, principalmente considerando as principais funcionalidades que espera-se ser disponibilizadas por ferramentas neste escopo. Foi possível observar e levantar diferentes formas de interação entre o indivíduo e as ferramentas desenvolvidas, sendo focado geralmente na simplicidade e na facilidade de uso. As informações aprendidas durante o estudo foram utilizadas para planejar quais funções são essenciais, quais devem ser adicionadas e principalmente como o usuário pode interagir com as mesmas.

Capítulo 4

Metodologia

Neste trabalho foi realizado o desenvolvimento de um *toolkit* para ser utilizado como apoio para a experimentação de sistemas BCI. Este *toolkit* tem como foco a produção da visualização do sinal de EEG em tempo real e permitir a aplicação de filtros espaciais e temporais no sinal obtido. Também é permitido a realização das demais fases de experimentação, bem como a extração de características e classificação dos dados.

Devido ao caráter prático do projeto proposto, foi utilizado uma metodologia inteiramente experimental, a qual consiste no desenvolvimento das ferramentas necessárias para os experimentos. O método de validação é realizado por meio de experimentos BCI utilizando as ferramentas desenvolvidas, sendo esperado que o *toolkit* possibilite a realização de experimentos semelhantes utilizando outras ferramentas. Ao longo deste capítulo, são apresentados as ferramentas e métodos escolhidos para o desenvolvimento do *toolkit* proposto.

4.1 Linguagens de programação utilizadas

Para o desenvolvimento deste trabalho, foram utilizadas duas linguagens de programação: `Python` e `Javascript`, sendo a primeira a linguagem principal adotada, enquanto a segunda foi utilizada para desenvolver a interface gráfica proposta, desenhada como um sistema Web para propiciar uma maior integração entre diferentes sistemas.

4.1.1 Python

Para o desenvolvimento dos módulos do *toolkit*, foi utilizado a linguagem de programação `Python`. A linguagem foi projetada durante o ano de 1999 ganhando grande popularidade entre a comunidade científica, estando atualmente entre as linguagens mais populares da atualidade [24]. A linguagem é uma das principais escolhas para projetos de análise de dados [3], devido a quantidade de bibliotecas científicas como: `Numpy` [29] e `matplotlib` [30], além de bibliotecas destinadas a projetos de inteligência artificial como a `Scikit-learn` [3] ou mesmo bibliotecas desenvolvidas para a realização de experimentos BCI como o `Gumpy` [22] e o `MNE-Python` [26].

`Python` é um projeto gratuito e de código livre o qual pode ser utilizado em muitas plataformas. Atualmente é utilizada como uma linguagem para uso geral tanto em âmbito acadêmico quanto em âmbito industrial [3].

4.1.2 Javascript

O `Javascript` é a principal linguagem de *script* para desenvolvimento Web, tendo sua popularidade aumentada após o surgimento do motor V8 [31]. Este motor foi desenvolvido inicialmente para o navegador Google Chrome e possibilita uma maior performance ao interpretar aplicações `Javascript`. Atualmente a linguagem não está limitada apenas ao desenvolvimento Web, podendo ser utilizada como uma linguagem de proposito geral, devido ao surgimento da ferramenta `nodejs`[1], a qual utiliza o motor V8 para interpretar códigos `Javascript` fora do contexto de navegadores de Internet. Para realizar o desenvolvimento da interface, foi utilizado o *framework* `vueJS`[2], que permite o desenvolvimento de interfaces Web baseadas em componentes, utilizando a linguagem `Javascript`.

[1] https://nodejs.org/
[2] https://vuejs.org/

4.1.3 Notação utilizada

Ao longo deste trabalho serão apresentados trechos de códigos em `Python` utilizando duas diferentes notações: (1) na forma de arquivos e (2) terminal iterativo. Na forma de arquivos, seguirão a sintaxe comum da linguagem utilizada para o desenvolvimento, como apresentada na Listagem 4.1.

Listagem 4.1: Exemplo da notação utilizada para arquivos na linguagem `Python`.

```
class Foo():
    def boo(self):
        pass
```

O Terminal iterativo será utilizado para alguns exemplos desenvolvidos em `Python` e apresentam a sintaxe utilizando ">>>"como um prefixo para indicar o início de um novo comando. Caso este comando produza algum tipo de saída, esta será exibida imediatamente na próxima linha, sem a presença de qualquer prefixo. Um exemplo da notação utilizada para o terminal iterativo é apresentado na Listagem 4.2.

Listagem 4.2: Exemplo da notação utilizada para o terminal iterativo presente na linguagem `Python`.

```
>>> print("terminal␣iterativo")
terminal iterativo
>>> 2+2
4
>>> a = 3
>>> a
3
```

4.2 Estrutura dos repositórios

A ferramenta foi dividida em dois repositórios principais a fim de separar partes e linguagens distintas de todo o projeto. O primeiro repositório é destinado para o núcleo do

toolkit[3], o qual está sendo desenvolvido utilizando a linguagem de programação `Python`. Enquanto o segundo repositório possui todo o código referente a implementação da interface gráfica[4], desenvolvida como uma aplicação WEB em `vueJS`.

4.2.1 Estrutura do projeto em Python

A ferramenta foi projetada desde o começo para permitir facilmente a expansão de seus módulos para suprir paradigmas não contemplados pelo projeto inicial. Foi utilizado esta abordagem devido a dificuldade de contemplar todas as possíveis soluções e paradigmas existentes para a realização de experimentos BCI. A estrutura do *toolkit* foi organizada em módulos específicos e independentes, permitindo a sua modificação dependendo da demanda do experimento, sem afetar o comportamento dos demais módulos. A organização dos módulos projetados esta apresentado na Figura 4.1, sendo as linhas tracejadas os módulos futuros já planejados.

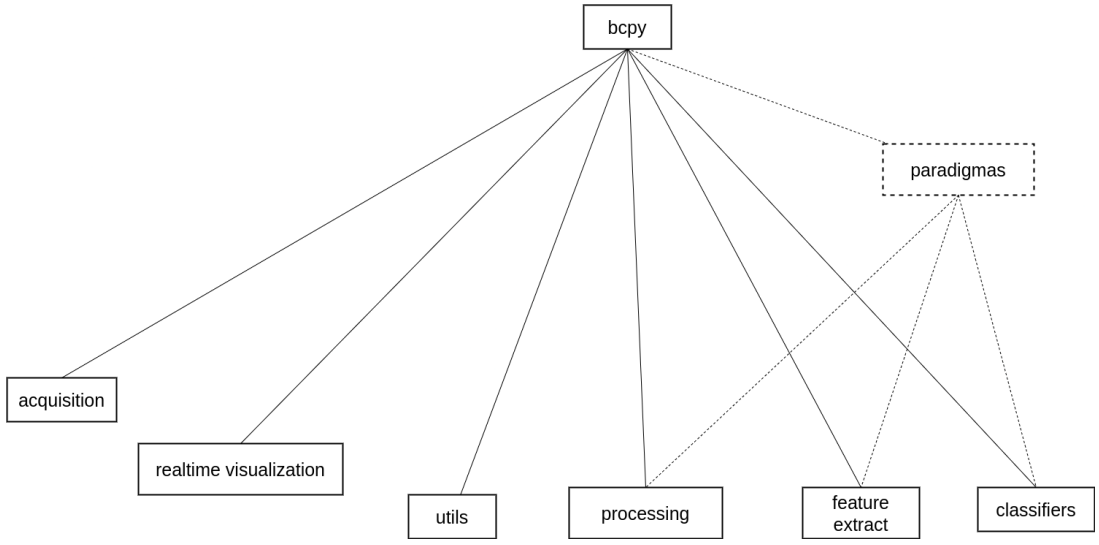

Figura 4.1: Organização dos módulos da biblioteca bcpy.

O *toolkit* foi projetado com módulos específicos para diferentes paradigmas, como forma de facilitar o desenvolvimento de novos experimentos. Estes módulos consistem em

[3]https://github.com/bneurd/bcpy
[4]https://github.com/bneurd/realtime-eeg-dashboard

versões pré-configuradas dos módulos já existentes, para assim abstrair parte da complexidade. Caso seja necessário, módulos gerais para todos os paradigmas podem ser utilizados como forma de complementar o experimento.

Para definir a estrutura utilizada para o projeto, foi realizado o levantamento de diversos repositórios de bibliotecas conhecidas em `Python`, para manter o projeto no mesmo padrão já conhecido pela comunidade de desenvolvedores da linguagem. As principais informações obtidas foram referentes aos repositórios `Gumpy`[5] e `Numpy`[6], sendo o primeiro uma biblioteca referente a sistemas BCI e o segundo a um conjunto de ferramentas para a manipulação de vetores e matrizes com padrões algébricos.

Foi observado que comumente é utilizado o uso de classes para o desenvolvimento, porém, essas são abstraídas do conhecimento do usuário final, os quais interagem apenas com funções simples disponibilizadas pela biblioteca. Tomando como base a biblioteca `Gumpy`, as classes implementadas são abstraídas do conhecimento do usuário devido a presença de uma função destinada a aplicar os métodos da classe, como apresentado na Listagem 4.3.

Listagem 4.3: Exemplo de organização de código do repositório `Gumpy`

```
class Foo():
    def boo(self):
        print("faz␣algo")

def apply_foo():
    Foo().boo()
```

Outro padrão comumente encontrado nos repositórios é a utilização de funções que não dependem de uma determinada implementação fixa, podendo ser alterado seu comportamento em tempo de execução, semelhantes ao padrão de projeto *strategy* [32]. Seguindo esse padrão, a implementação dos determinados algoritmos não estão presos a classe principal, podendo ser trocados por outros já existentes em tempo de execução. Isso facilita a

[5]https://github.com/gumpy-bci/gumpy
[6]https://github.com/numpy/numpy

implementação de novos algoritmos sem alterar o comportamento da classe principal. O diagrama de classes em formato da Linguagem de Modelagem Unificada (do ingles, *Unified Modeling Language* (UML)) do padrão de projeto *Strategy* é apresentado na Figura 4.2.

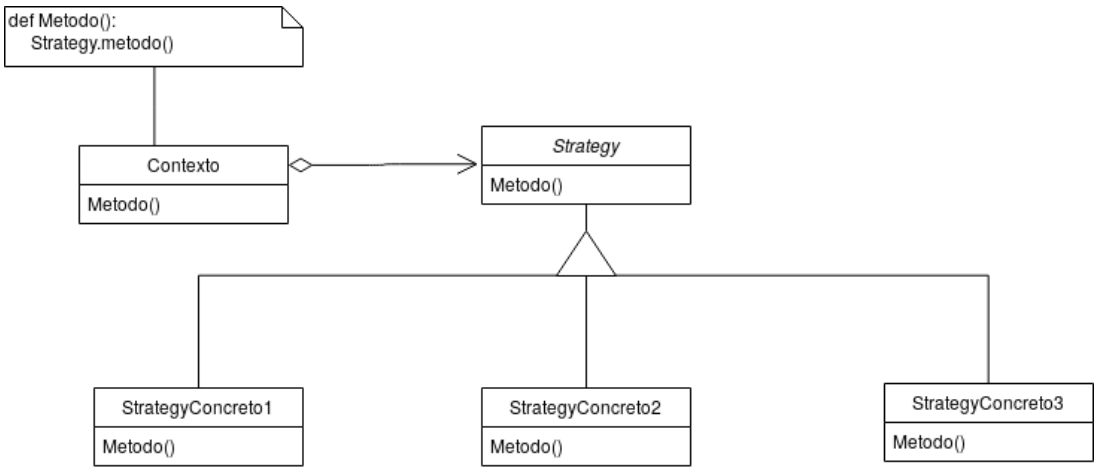

Figura 4.2: Diagrama de classes representando o padrão de projeto *Strategy* [32].

A utilização do padrão *strategy* pelas bibliotecas levantadas é realizada de forma diferente da proposta original. Segundo os repositórios levantados, para possibilitar a troca do comportamento de uma função é utilizado uma `string`, a qual define o comportamento da classe, como apresentado na Listagem 4.4. Isto possibilita que uma mesma função, ao receber uma `string` diferente, obtenha um novo comportamento.

Listagem 4.4: Exemplo da definição de um comportamento de uma função em tempo de execução utilizando uma `string`.

```
>>> foo("tipo1")
comportamento tipo 1
>>> foo("tipo2")
comportamento tipo 2
```

A implementação utilizada do padrão *strategy*, seguindo os padrões levantados, está apresentada na Listagem 4.5 e foi realizada utilizando uma estrutura de chave e valor presente na linguagem `Python` chamada de dicionário, na qual a chave é a `string` que representa o comportamento a ser utilizado pela função e o valor, a classe que implementa

o comportamento. Desta forma é realizado um comportamento semelhante a um método fábrica [32], o que evita a utilização de estruturas condicionais para instanciar novos objetos.

Listagem 4.5: Exemplo de instanciação de um objeto utilizando um dicionario de Strings.

```
class Foo():
    def boo(self):
        print("faz␣algo")

ref_classes = {"Foo": Foo};

def apply_foo(cls_str):
    # instancia um novo objeto com base em uma string.
    ref_classes[cls_str]().boo()
```

Cada módulo implementado possui três elementos principais:

1. **classe abstrata**: define a interface dos métodos;
2. **classes concretas**: implementações da classe abstrata;
3. **função principal**: responsável por abstrair e executar os métodos das classes concreta.

4.2.2 Estrutura do projeto em vueJS

O segundo repositório que foi desenvolvido é referente a interface gráfica Web, desenvolvida em `Javascript` utilizando a ferramenta `vueJS`. A aplicação possui uma arquitetura cliente-servidor, na qual a comunicação ocorre por meio de um *websocket* [33]. A comunicação ocorre entre a interface gráfica (cliente) e o módulo de visualização de dados (servidor) disponibilizado no *toolkit* que está sendo desenvolvido.

4.3 Bases de dados pública

Durante a etapa de desenvolvimento, foi utilizado bases de dados públicas disponibilizadas gratuitamente. Nesta seção é apresentado a base de dados AVI SSVEP [34] e a base EEG Database Data Set [35].

4.3.1 Descrição da base de dados AVI SSVEP

A base de dados AVI SSVEP [34] contém dados EEG mensurados de indivíduos saudáveis ao serem expostos a alvos piscantes em um monitor LCD com taxa de atualização de 120 Hz para evocar um sinal SSVEP. Todos os experimentos foram gravados utilizando 3 elétrodos (Oz, Fpz e Pz), posicionados seguindo o sistema internacional 10-20, sendo os dados atribuídos ao elétrodo Oz, referente a região ocipital do cérebro, os únicos gravados na base de dados.

Os participantes foram posicionados a 60 cm do monitor LCD sendo que cada estímulo possuía uma área de 2,89 cm^2. O fluxo do experimento está demonstrado na Figura 4.3.

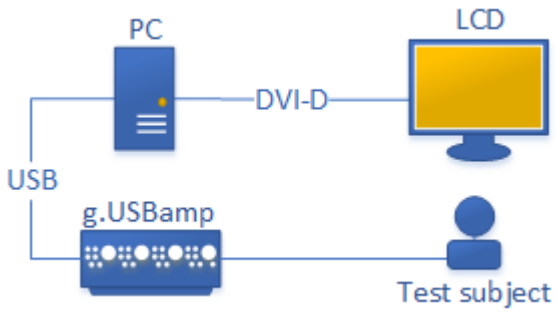

Figura 4.3: Fluxo do Experimento [34]

Em todos os dados foram aplicados, como forma de pré-processamento, um filtro *notch* nas frequências em torno de 50 Hz com o intuito de remover a interferência causada pela rede elétrica (na Europa).

Para compor a base de dados, foram desempenhados dois experimentos. O primeiro foi utilizando apenas um alvo, em que quatro participantes realizaram uma única sessão

com 4 ensaios. Em cada ensaio, os participantes focaram no único alvo piscante durante 30 segundos.

O segundo experimento possuía 7 alvos cintilantes, em diferentes frequências, cada alvo está separado dos demais em uma distância horizontal e vertical de 1.8 cm e 2.1 cm respectivamente, como apresentado na Figura 4.4. Foram realizadas duas sessões de experimento com cinco participantes cada. Em cada sessão, os participantes passaram por 10 ensaios focando em um determinado alvo por 16 segundos.

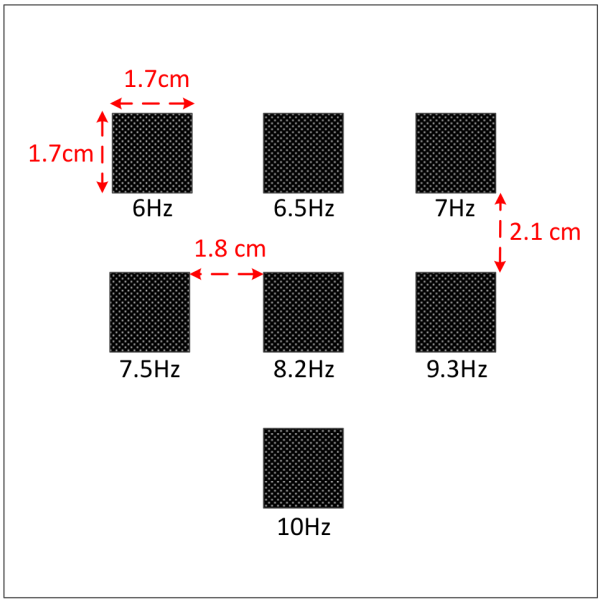

Figura 4.4: Disposição dos alvos cintilantes durante o segundo experimento [34].

4.3.2 Descrição da base de dados EEG Database Data Set

Na base de dados *EEG Database Data Set* [35] são armazenados dados EEG obtidos de um estudo com o intuito de correlacionar uma predisposição genética para o alcoolismo. Foram armazenados dados referentes a 64 elétrodos gravados em uma frequência de 256 Hz, com 1 segundo de duração.

Dois grupos de indivíduos foram utilizados para compor o experimento, um grupo de controle e um de alcoólatras, totalizando 122 indivíduos, sendo que cada sujeito realizou 120 sessões com diferentes estímulos.

Esta base de dados foi disponibilizada em 3 versões: pequena; grande; e completa, sendo que cada uma diferencia-se apenas na quantidade de dados disponíveis. Para a realização deste projeto, foi utilizado a base de dados grande.

4.3.3 Simulação dos dados de EEG

Para facilitar o desenvolvimento e possibilitar a realização do trabalho proposto sem a dependência do hardware específico foram simulados experimentos por meio de dados de EEG. Para isto, foram utilizados os dados obtidos das bases de dados públicas, para assim ser possível testar as funcionalidades utilizando dados reais de experimentos conhecidos.

Foi desenvolvido uma aplicação para a leitura de cada base de dados e a transmissão dos dados utilizando o protocolo `LSL`, com o auxílio da biblioteca `pylsl`[7], uma implementação do protocolo `LSL` para `Python`.

4.4 Considerações finais

Neste capítulo foi apresentado a metodologia que foi utilizada para o desenvolvimento deste projeto, sendo focada principalmente em como será realizado o processo de desenvolvimento do *toolkit* proposto. Para realizar a validação do *toolkit*, foi realizado experimentos completos utilizando o conjunto de ferramentas BCPY desenvolvido. Após isto, foi possível verificar a capacidade do *toolkit* em realizar todas as etapas de um experimento. Foi utilizado principalmente para testes experimentos de atenção seletiva, devido o maior conhecimento prévio sobre o paradigma e por utilizar todas as etapas clássicas de um experimento BCI, outros experimentos com diferentes variações das etapas também foram utilizadas.

[7] https://github.com/chkothe/pylsl

Capítulo 5

Projeto

Este capítulo apresenta o desenvolvimento do *toolkit* proposto. O *toolkit* consiste em um conjunto de ferramentas destinadas a facilitar o desenvolvimento de experimentos BCI em tempo real, ou seja, aplicações em que o processamento dos dados ocorre logo em sequência em relação a aquisição dos dados, devolvendo uma resposta ou ação do sistema referente a intenção do usuário. O *toolkit* foi desenvolvido pensando em um fluxo principal, referente ao fluxo comum de uma aplicação BCI, em que um indivíduo pode simplesmente adicionar e remover módulos referentes a cada ação necessária para o experimento.

O fluxo principal projetado para o *toolkit* é apresentado na Figura 5.1. Além da entrada e saída de cada módulo a qual pode ser utilizada para montar o experimento da maneira necessária, modificando a ordem ou simplesmente removendo ou adicionando módulos específicos para cada tipo de experimento realizado. Os diferentes módulos têm como entradas básicas iteradores da linguagem `Python`.

Os iteradores foram construídos utilizando o conceito de geradores[1], que permitem a construção de funções com o comportamento de iteradores. A principal vantagem dos geradores é a possibilidade de utilizar os valores sob demanda, sem a necessidade de esperar a execução completa da função para obter os valores já gerados. A utilização de funções geradoras são realizadas por meio do comando `yield`, como apresentado na Listagem 5.1, sendo que cada vez que a função `next` é invocada, a função `generate_vet` é

[1] https://wiki.python.org/moin/Generators

executada até encontrar um comando `yield`. Quando isto ocorre, a função retorna o valor atual e bloqueia a sua execução até o próximo `next` ser executado. É importante salientar que apenas o valor atual é salvo em memória, desta forma, o valor atual é sobrescrito a cada invocação da função `next`.

Listagem 5.1: Exemplo de funções geradoras na linguagem `python`

```
>>> def generate_vet(size):
...     for i in range(size):
...         yield(size)

>>> iterator = generate_vet(3)
>>> next(iterator)
0
>>> next(iterator)
1
```

Neste *toolkit*, os iteradores podem representar tanto um único dado, quanto um conjunto denominado de *buffers*. Qualquer iterador gerado por um módulo pode ser utilizado como entrada de outro que aceite este tipo de entrada. O módulo de visualização dos dados em tempo real pode aceitar tanto um iterador de dados simples quanto um iterador de *buffers*, possibilitando o usuário escolher o melhor ponto para visualizar os dados. É importante ressaltar que apenas um único módulo de visualização pode ser utilizado por experimento, pois este inicia um servidor Socket interno para transmitir os dados, sendo a utilização de mais iria resultar em um conflito dos dados enviados.

A seguir, apresenta-se o desenvolvimento dos módulos do *toolkit*, descrevendo os detalhes de implementação e decisões estruturais tomadas para o projeto.

5.1 Aquisição

O módulo de aquisição é responsável por obter os dados mensurados pelo equipamento de EEG. Para isto, inicialmente foi implementado esta funcionalidade utilizando o protocolo

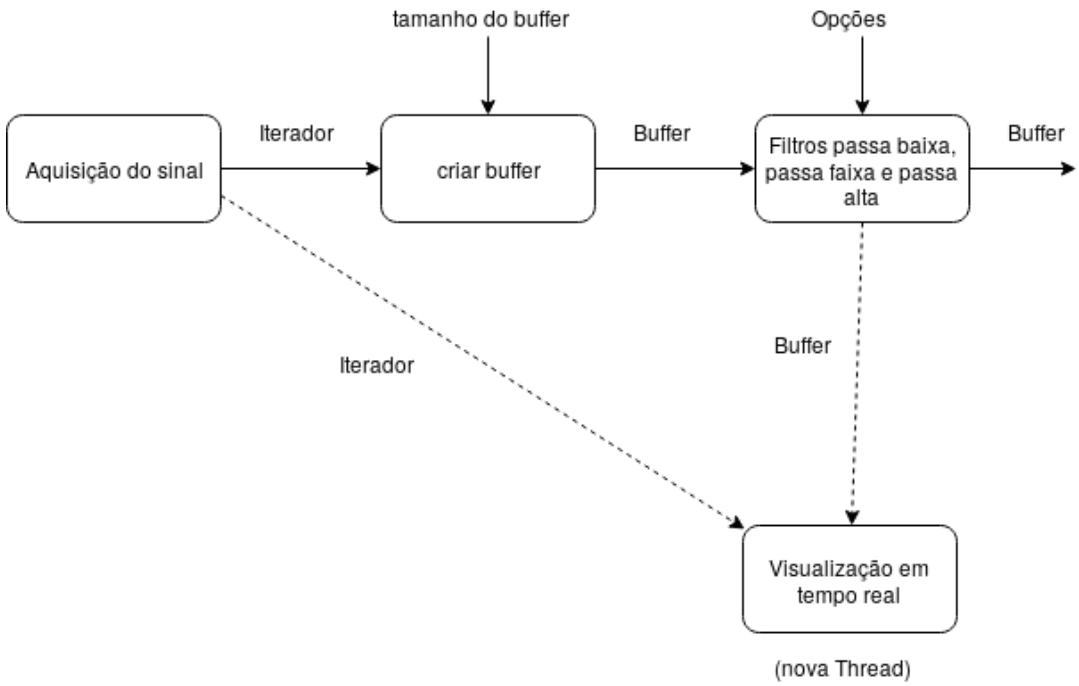

Figura 5.1: Fluxo básico da biblioteca bcpy.

de transmissão `LSL`, comumente utilizado para aplicações desta área. A escolha desta estratégia foi devido a simplicidade da implementação e por obter uma solução genérica o suficiente para abranger grande parte dos hardwares disponíveis no mercado para a captura do sinal. A Listagem 5.2 apresenta como foi definido a classe abstrata para a aquisição dos dados na biblioteca.

Listagem 5.2: Classe abstrata base para a aquisição de dados

```
class Acquisition(ABC):
    @abstractmethod
    def get_data(self):
        yield(data)

    @abstractmethod
    def get_data_label(self):
        yield(data)
```

```
@abstractmethod
def terminate(self):
    pass
```

A interface disponível para este módulo disponibiliza a função `get_data`. Esta recebe como parâmetro um objeto filho da classe de aquisição ou o nome da estratégia escolhido para realizar a aquisição do sinal. Além disto, pode ser passado alguns argumentos opcionais de configuração de cada estratégia. As estratégias disponíveis estão apresentadas na lista a seguir.

- `LSL`: Estratégia de aquisição do sinal utilizando o protocolo LSL.
- `Custom`: Estratégia de aquisição do sinal personalizada.

É importante notar que a função `get_data` tem como retorno um iterador, isto significa que este pode ser utilizado para percorrer todos os dados obtidos e servir de entrada direta para outros módulos presentes na biblioteca.

Além da função `get_data`, o *toolkit* disponibiliza a função `get_data_label`, a qual apresenta um funcionamento semelhante, porem permitindo adicionar uma estratégia para obter as *labels* de cada porção de dados. A função `get_data_label` permite utilizar as mesmas estratégias de aquisição do sinal que a função `get_data`.

5.2 Utilidades

O *toolkit* provê um módulo referente a utilidades necessárias para suas operações. Atualmente é disponibilizada uma função destinada a converter um iterador simples, de um único dado, em um iterador de *buffers* de tamanho `N`. Uma função destinada a consumir os iteradores também foi implementada, visto que esta é uma operação necessária para o funcionamento do *toolkit*, devido as características dos geradores da linguagem `Python`. Para comportar diferentes ações necessárias para os mais variados tipos de experimentos, é permitido desenvolver uma função customizada, a qual será executada para cada

iteração. A Listagem 5.3 apresenta a utilização de uma função customizada para obter o maior ritmo de frequência.

Listagem 5.3: Exemplo de função customizada

```
...
def process(psd_value):
    f, psd_values = psd_value
    bands = [0, 0, 0, 0]
    bands[0] = band_extract(f, psd_values, 8, 12)
    bands[1] = band_extract(f, psd_values, 5, 7)
    bands[2] = band_extract(f, psd_values, 12, 30)
    bands[3] = band_extract(f, psd_values, 25, 100)

    maxIndex = np.argmax(bands)
    print(maxIndex)


flow(psd_buff, function=process)
```

Após a transformação de um único dado em um conjunto (*buffer*) o *toolkit* disponibiliza uma função destinada a transformar os dados simples em um objeto esperado pelas demais partes do fluxo. Atualmente, essa funcionalidade encapsula, removendo toda a complexidade, a criação de um objeto `Raw` da biblioteca `mne-python`, sendo necessário a apenas informar, alem do gerador dos dados, a frequência dos dados e uma lista contendo os nomes dos canais presentes na gravação (devendo estar inclusive na mesma ordem dos dados). A Listagem 5.4 apresenta o funcionamento da função para criar os objetos EEG a partir de um gerador de buffer.

Listagem 5.4: Exemplo de utilização da função `create_eeg_object`

```
fs=256
objsGen = create_eeg_object(BufferGen, fs, channels=['o1', 'o2])
```

Uma função destinada a salvar os valores obtido em um arquivo também foi desenvolvida, permitindo atualmente o armazenamento dos dado em um arquivo no formato `JSON`.

5.3 Exibição dos dados em tempo real

Para um controle maior durante a aquisição dos dados, foi desenvolvido um módulo destinado a exibir em tempo real os dados obtidos. Desta forma, durante o desenvolvimento do experimento é possível obter um *feedback* dos componentes e certificar que tudo está funcionando como o esperado. Este módulo pode receber como entrada de dados, tanto iteradores de dados únicos, quanto iteradores com dados em *buffer*, ou até mesmo um objeto `Raw`. Assim é possível encaixar este módulo em qualquer etapa do experimento, permitindo o indivíduo escolher em que momento será mais apropriado para cada uso. Para o módulo de visualização não proporcionar um atraso no processamento dos dados, sua execução ocorre em uma nova *thread*, permitindo que o programa continue seu fluxo independentemente da visualização.

O módulo de visualização em tempo real foi concebido para suportar a expansão para diferentes formas de exibição e em diferentes interfaces. Atualmente existe uma forma de visualização dos dados em uma página Web, permitindo a sua utilização em diferentes sistemas operacionais, além de uma mesma versão utilizando um método de *Webview*, o qual utiliza menos recursos do que um navegador Web.

5.3.1 Exibição dos dados em um navegador Web

Foi projetado a exibição de dados como uma página web para evitar qualquer problema de compatibilidade entre sistemas operacionais, tornando assim, o *toolkit* capaz de ser utilizada de forma eficiente em diferentes plataformas. Para manter o servidor web executando, foi utilizado o *framework* `Flask`[2]. Em conjunto, foi utilizado um *websocket* para

[2]http://flask.pocoo.org/

realizar a comunicação dos dados e a página Web. Desta forma, os dados adquiridos são enviados para o servidor, o qual transmite para a página Web.

Para realizar a exibição dos dados em tempo real, foi realizado um estudo empírico para selecionar a melhor ferramenta `javascript` para este fim. Foi selecionada a biblioteca `Smoothie Charts`[3], que disponibiliza funções para criação de gráficos de forma fácil e eficiente. Foi utilizada também a biblioteca Chart.js[4] para a criação dos demais gráficos, os quais não necessitam de uma taxa de atualização muito alta.

5.3.2 Exibição dos dados em um Webview

Com a intenção de diminuir a quantidade de recursos gastos pela aplicação, focando futuramente no processamento dos dados online, foi adicionado a opção de exibir a visualização dos dados em uma janela Webview. Esta funcionalidade foi desenvolvida utilizando o *framework* `Electron`[5] e reproduz de forma equivalente a aplicação Web para um aplicativo híbrido para Desktop. Deste modo, a visualização mantém sua característica de ser multiplataforma, porém, sem dividir seu processamento com outras funções de um navegador Web. Esta estratégia é usado comumente em um ambiente `mobile`, para melhorar o desempenho comparado a um navegador e mantém a facilidade de desenvolvimento [36].

5.3.3 Interface gráfica

A interface implementada busca manter a simplicidade focando as funcionalidades mais utilizadas. A aplicação exibe de forma independente os dados obtidos de cada elétrodo, uma exibição da densidade espectral de todos os elétrodos e medições de ondas comuns para sistemas BCI, sendo estas os ritmos alfa, beta, delta, gama e teta. A interface implementada é apresentada na Figura 5.2 durante a execução de um experimento. Deste modo, a interface exibe indicadores do estado da sua conexão entre a aplicação principal, a frequência de transmissão e os dados mensurados pelo EEG.

[3]http://smoothiecharts.org/
[4]https://www.chartjs.org/
[5]https://electronjs.org/

Figura 5.2: Interface de aquisição dos dados.

Inicialmente, foi escolhido manter um gráfico para cada canal de elétrodos, porém, seguindo esta abordagem, foi encontrado dificuldades durante a visualização dos dados no domínio do tempo, pois cada gráfico apresentava sua própria escala, sendo impossível definir uma escala genérica o suficiente que supra todos os experimentos, causando uma falsa impressão ao analisar os dados exibidos.

Para solucionar este problema foi decidido manter apenas um gráfico para todos dados, permitindo assim a comparação de cada dado em uma escala única. Para isto, o eixo Y é divido em N porções, sendo que cada canal utiliza de duas porções para exibir seus dados.

Apesar da delimitação de espaço para cada elétrodo no gráfico, nada impede que eventualmente, os valores obtidos ultrapassem suas porções e adentrem a área de outro elétrodo, porém não se trata de um problema, sendo esta a solução mais utilizada para dados neste domínio.

Para realizar a unificação dos gráficos, foi necessário deslocar os valores de cada linha a fim de separá-las pelo eixo Y, permitindo uma melhor distinção entre cada linha. Para calcular este deslocamento, foi aplicado a Fórmula 5.1 sobre os dados em volts. Note que

a fórmula do deslocamento centraliza os dados dentro do espaço disponível, ao subtrair uma unidade do valor obtido.

$$2 * (numChannelsTotal - numChannel) - 1 \tag{5.1}$$

O plot dos dados no domínio da frequência foi realizado com o auxílio de um janelamento de 3 segundos. Isto significa que inicialmente, a interface apenas acumula dados de três segundos antes de começar a realizar seu processamento. Após 3 segundos, a janela é deslocada e processada a cada 0.25 segundos. Esta abordagem garante a existência de uma quantidade de pontos consideráveis para permitir a transformada do domínio do sinal. Um exemplo de janelamento de 3 unidades e deslocamento de uma unidade é apresentado na Figura 5.3.

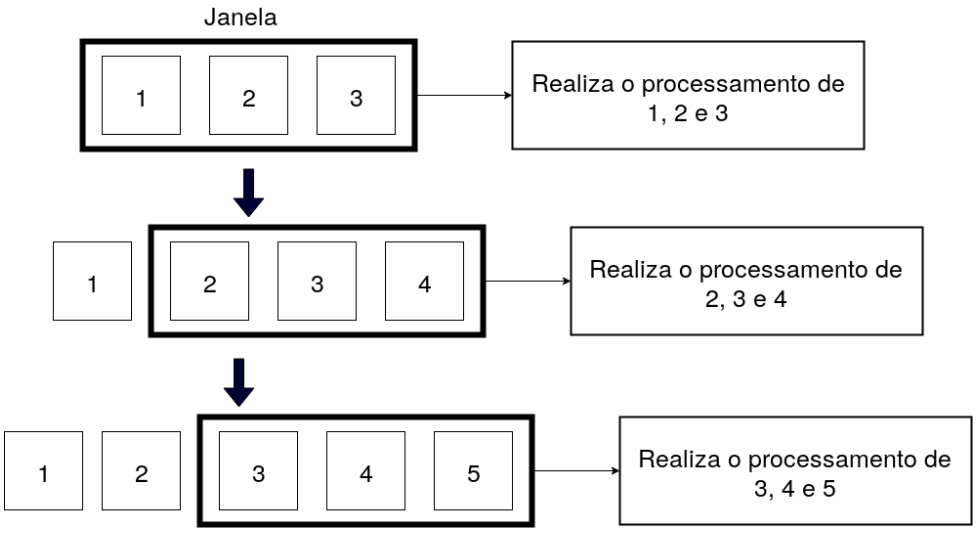

Figura 5.3: Demonstração da aplicação de um janelamento de 3 unidades com deslocamento de uma unidade.

5.4 Processamento dos dados

O módulo de processamento dos dados tem a função de aplicar filtros em tempo real durante o experimento. Os filtros presentes dentro deste módulo possuem como entrada

básica um *buffer* dos dados brutos e devolvem como saída um novo *buffer* com os dados filtrados. Foram implementadas duas funções principais: uma destinada a aplicar filtros temporais em geral e a outra com o intuito de aplicar o filtro *notch*.

Para aplicar um filtro passa-faixa, passa-baixa ou passa-alta, deve ser utilizado a função `apply_filter`. Esta função recebe três argumentos: 1) os dados, (2) um valor de corte inferior e (3) um valor de corte superior. Desta forma, caso seja passado apenas um valor de corte superior ou inferior, a função aplicará um filtro passa-baixa ou passa-alta, respectivamente. Mas, caso os dois valores sejam informados, será aplicado um filtro passa-faixa aos dados. Filtros *notch* também podem ser aplicados no sinal em *buffer*, mas para isto deve ser utilizado a função *notch* provinda do módulo de processamento. Uma comparação dos dados no domínio da frequência antes e depois da aplicação de um filtro passa faixa entre 5 Hz e 40 Hz está apresentada na Figura 5.4, no qual o gráfico da esquerda apresenta um ruído maior presente nas frequências menores que 5, enquanto o mesmo é amenizado no gráfico da direita.

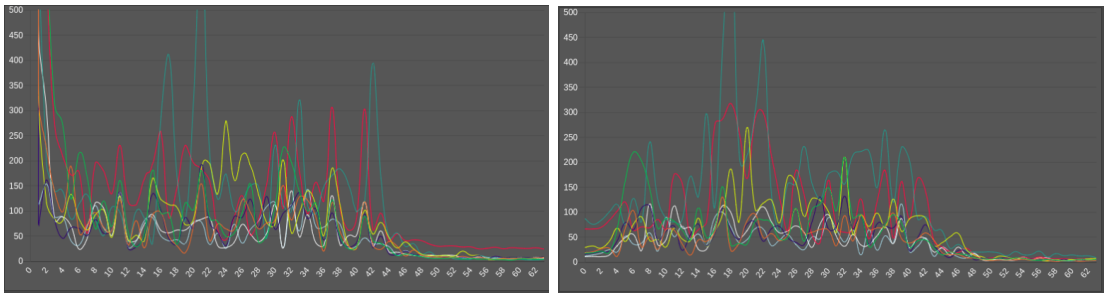

Figura 5.4: Esquerda: dados sem a aplicação de filtros; Direita: dados apos a aplicação de um filtro passa faixa de 5 a 40 Hz.

Além dos filtros temporais citados, foram desenvolvidos 2 filtros espaciais: `car` e `drop_channel`. A assinatura das duas funções desenvolvidas são semelhantes, recebendo 2 argumentos: 1) o iterador de objetos e 2) uma lista com os nomes dos canais. A principal diferença entre as funções é dada pelo comportamento, que enquanto na função `drop_channel` apenas ira remover os canais especificados, a função `car` ira aplicar um algoritmo para atenuar os dados presentes nos canais especificados em relação aos demais canais.

5.5 Extração de características

O módulo de extração de características possui um comportamento diferente dos demais módulos do *toolkit* pois para evitar a duplicação da chamada da função de extrair características, foi optado por unificar estas chamadas e controlar isto internamente, restando ao usuário apenas configurar as opções de cada chamada. Isto ocorre para permitir a extração de diferentes tipos de características de forma simultânea.

A assinatura da função de extração de características é dada com dois argumentos: 1) Um iterador de objetos e 2) uma lista de extração de características. A função então ira aplicar em ordem as estratégias especificadas para cada objeto iterado em sua execução. e terá como retorno um novo iterador de listas uni-dimensionais contendo as características extraídas.

Atualmente, o *toolkit* suporta dois tipos de estratégias para a extração de características, sendo este a extração do PSD do sinal e a extração de faixas de frequências especificadas. A Listagem 5.5 apresenta um exemplo de funcionamento da função de extração de características utilizando as duas estratégias disponíveis de forma simultânea.

Listagem 5.5: Exemplo de função customizada

```
...
strategies = [
    BandExtract(['alpha', 'beta', 'gamma', 'delta', 'theta']),
    PSD()
]

features = extract(objsBuffer, strategies)
```

5.6 Classificadores

O *toolkit* desenvolvido possibilita a utilização de algoritmos de classificação de forma fácil. Diferente dos outros módulos, a etapa de classificação possui algumas peculiaridades para

suportar experimentos online. Desta forma, a etapa de classificação deve ser separada em duas partes, sendo uma destinada a treinar o classificador, e a outra para a sua utilização.

Para manter a simplicidade, as duas etapas necessárias para o modulo de classificação podem seguir exatamente o mesmo fluxo, sem a necessidade de fazer alterações que não são relacionadas ao módulo de classificação. As duas etapas podem ser executadas em diferentes momentos, não sendo necessárias de ser executadas em sequência, permitindo com que o pesquisador possa ter tempo para realizar ajustes e modificações no experimento.

O função do módulo de classificação é encapsular a complexidade de algoritmos conhecidos de aprendizagem de máquina, retirando a complexidade da mão de pesquisadores menos experientes. Dentro deste módulo não são implementados manualmente os classificadores, visto que estes já estão disponíveis e devidamente optimizados. Os classificadores disponibilizados nesta tese foram obtidos utilizando a biblioteca `Scikit-learn`[6].

5.6.1 Etapa de treinamento

Durante a etapa do treinamento do classificador, o usuário é capaz de escolher entre diferentes algoritmos de classificação disponíveis pelo *toolkit*, treinar o classificador e salvar o classificador treinado em um arquivo binário.

Atualmente 3 algoritmos de classificação estão disponíveis pelo *toolkit* sendo estes: (1) KNN; (2) SVM; e (3) LDA. A seleção de algoritmos foram escolhidos de modo a suprir as necessidades dos experimentos utilizados para teste, porém esses algoritmos podem ser utilizados para demais tipos de experimento. A adição de novos algoritmos de classificação pode também ser feita de maneira simples, similar aos demais módulos.

5.6.2 Etapa de classificação

Durante a etapa de classificação, o pesquisador deve carregar o modulo treinado para realizar as predições no experimento. O fluxo do experimento se mantém o mesmo, com

[6]https://scikit-learn.org/stable/index.html

exceção do modulo de classificação em si. Ao carregar um modelo treinado, o *toolkit* retorna uma instância de um objeto de classificador, este objeto funciona para encapsular e padronizar os métodos disponíveis pelos diferentes classificadores. Atualmente, este objeto disponibiliza funções para realizar a predição de valores e para calcular a taxa de acerto do classificador.

5.7 Considerações finais

Durante o desenvolvimento deste trabalho foi possível realizar o desenvolvimento dos módulos de visualização e de processamento de sinais, estes módulos possuem um carácter genérico, permitindo serem utilizados em qualquer paradigma. Sendo esta a parte fundamental para futuras alterações, as quais podem possibilitar a expansão do *toolkit* para paradigmas diferentes e mais específicos.

Capítulo 6

Resultados e Discussão

Este capítulo apresenta os testes realizados para verificar que o projeto desenvolvido cumpre os objetivos propostos, permitindo a realização de experimentos BCI. Neste capitulo são listados 3 projetos de complexidade e métodos de execução diferentes, com o objetivo de tentar contemplar uma maior variedades de tipos de experimentos.

6.1 Mensuração dos ritmos alfa

O primeiro teste implementado apresenta um fluxo simplificado de um experimento BCI, no qual o objetivo principal é identificar quando os ritmos alfa se encontram superiores aos demais ritmos. Apesar da simplicidade do problema e de sua resolução, há grandes utilidades para este escopo, como o desenvolvimento de sistemas capazes de identificar, apenas olhando para os ritmos, indivíduos sonolentos, focados (alterando a analise para ritmos "beta"), entre outros. São inúmeras aplicações possíveis de serem desenvolvidas tendo a informação da potência de um ritmo e o estado do indivíduo.

Para realizar a mensuração dos valores do ritmo alfa, não é necessário utilizar classificadores. Para manter a simplicidade deste primeiro teste, foi escolhido realizar apenas uma análise simples nos dados obtidos de um PSD. A Figura 6.1 apresenta os passos necessários para realizar a implementação do experimento.

Os passos necessários para o experimento são os clássicos de um experimento BCI,

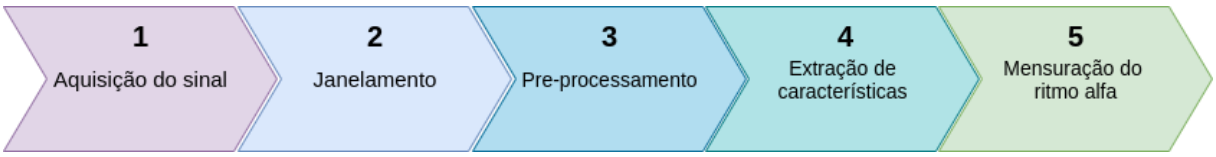

Figura 6.1: Passos necessários para realizar o experimento de mensurar o valor dos ritmos alfa.

substituindo apenas o uso de um classificador robusto por uma comparação manual entre os ritmos. Para a realizar a implementação, foi utilizado as funções disponibilizadas pelo *toolkit* BCPY, o qual disponibiliza métodos simples para realizar os primeiros 4 passos, tornando necessário programar apenas o método de comparação entre os ritmos. Para este experimento, foi utilizado a aquisição do sinal por meio do protocolo LSL, simulando o que ocorreria em uma aplicação BCI *online*. Foi utilizado uma janela de 3 segundos com uma intersecção de 2 segundos e aplicado o filtro passa-faixa 3 vezes, com o objetivo de intensificar a amenização das frequências filtradas. Por fim, foi extraído como características as frequências dos ritmos comparados.

Para realizar a comparação dos ritmos, foi necessário implementar um procedimento manual. Da forma implementada, caso o ritmo alfa seja maior do que os demais, é exibido um valor entre 1 e 100 referente a superioridade entre o ritmo alfa e os demais ritmos, caso contrario, apenas o valor 0 é exibido. A Listagem 6.1 apresenta o código desenvolvido para realizar a mensuração dos ritmos alfa.

Listagem 6.1: Experimento desenvolvido para mensuração de ritmos alfa.

```
...

data = getdata("LSL")
buff_win = create_window(data, 768, 512)
eeg_obj = create_eeg_object(buff_win, fs=250, channels=[
                            "1", "2", "3", "4", "5", "6", "7", "8"])
filter_buff1 = bandfilter(eeg_obj, lo=5, hi=50)
filter_buff2 = bandfilter(filter_buff1, lo=5, hi=50)
filter_buff3 = bandfilter(filter_buff2, lo=5, hi=50)
```

```
features = extract(filter_buff3, [BandExtract(
    ['alpha', 'beta', 'gamma', 'delta', 'theta'], average=True)])

def scala100(maxValue, minValue):
    return minValue*100/maxValue

def process(feature):
    maxIndex = np.argmax(feature)
    if (maxIndex == 0):
        secondMaxIndex = np.argmax(feature[1:])+1
        maxValue = feature[maxIndex]
        minValue = feature[secondMaxIndex]
        print(scala100(maxValue, minValue))
    else:
        print(0)

flow(features, function=process)
```

A Figura 6.2 apresenta o gráfico gerado pela interface gráfica desenvolvida comparando os ritmos em determinado momento do experimento.

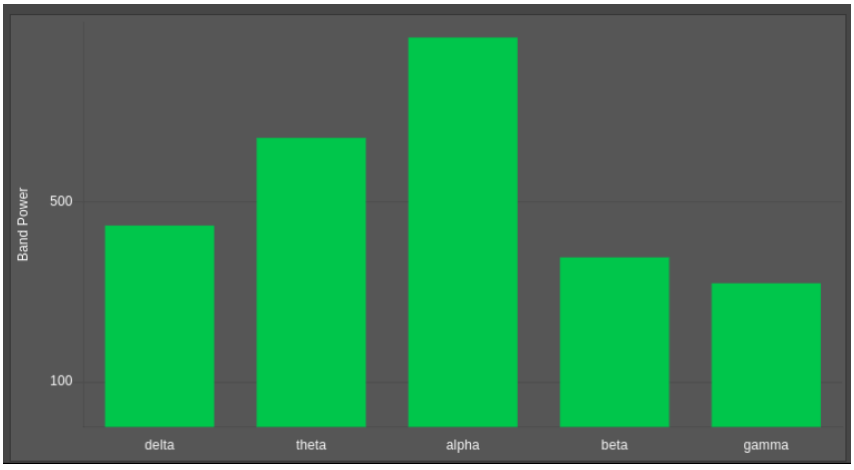

Figura 6.2: Comparação entre os ritmos.

6.2 Experimento SSVEP

O segundo experimento realizado utiliza o paradigma SSVEP com a base de dados `AVI SSVEP`. O experimento consiste em processar os dados e classificar de forma correta qual estímulo estava sendo aplicado para cada porção dos dados.

6.2.1 Fluxo do experimento

O experimento foi dividido em duas etapas, sendo a primeira destinada a treinar o modelo de classificador, enquanto a segunda apenas utiliza o classificador já treinado para predizer o estímulo. É importante ressaltar que as duas etapas podem ser executadas em momentos distintos, sem a obrigatoriedade de serem executadas juntas.

Como cada indivíduo presente na base de dados AVI SSVEP foi submetido a duas seções diferentes, cada uma destas foi utilizado para uma das duas etapas, treinamento do classificador e validação do classificador.

Devido a característica da base de dados utilizada, os dados adquiridos já estavam corretamente divididos, tirando a necessidade de adicionar uma etapa de janelamento no fluxo desenvolvido.

A Figura 6.3 apresenta o fluxo criado para a etapa de treinamento do experimento. Nesta etapa, os dados obtidos passam por uma filtragem passa-faixa dos valores entre 5 e 50, com o objetivo de remover ruídos no sinal. Após a filtragem dos dados as características são extraídas e usados como entrada para o treinamento do classificador. Ao final do fluxo, o modelo de classificador treinado é salvo para que possa ser utilizado posteriormente. As características extraídas e utilizadas no experimento foram os valores no domínio das frequências evocadas, com a intenção do classificador aprender que, como diz o paradigma SSVEP, que o maior valor corresponde a frequência evocada.

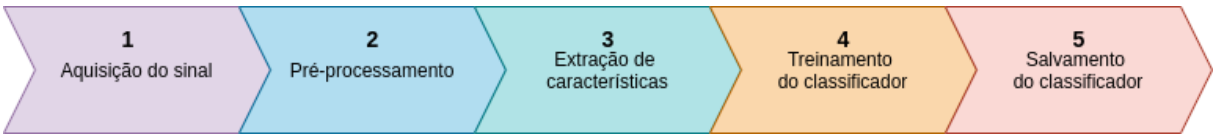

Figura 6.3: Passos necessários para realizar a etapa de treinamento do experimento.

A segunda etapa do experimento segue bastante semelhante a etapa de treinamento, diferenciando apenas na etapa de classificação dos dados obtidos. A Figura 6.4 apresenta o fluxo completo da etapa de validação e classificação do experimento.

Figura 6.4: Passos necessários para realizar a etapa de classificação do experimento.

Durante a realização do experimento, foi utilizado o classificador *Support-Vector Machines* (SVM) com o *kernel* linear devido sua simplicidade e bons resultados para experimentos dentro do paradigma SSVEP. A Listagem 6.2 apresenta de forma resumida o código desenvolvido para realizar o experimento em questão, apresentando apenas a etapa de utilizada para treinar o modelo de classificação. O código na sua integra pode ser visualizado no repositório disponível no Github[1].

Listagem 6.2: Experimento desenvolvido para mensuração de ritmos alfa.

```
...

data, labels = getdata_label(
    "Custom", get_data=get_custom_data, get_label=get_custom_label)
objs = create_eeg_object(data, 256, ["o2"])
filter_buff1 = bandfilter(objs, lo=5, hi=50)
filter_buff2 = bandfilter(filter_buff1, lo=5, hi=50)
filter_buff3 = bandfilter(filter_buff2, lo=5, hi=50)
features = extract(filter_buff3, [BandExtract([6, 7, 7.5, 8.2], 0.5)])
classifier = training('SVM', features, labels, verbose=True)
classifier.save('model.joblib')
```

[1]https://github.com/bneurd/bcpy/blob/master/teste_projeto2.py

6.2.2 Dificuldades presentes no experimento

Foram encontrados ao longo da realização dos experimentos, algumas dificuldades que foram necessárias de levar em consideração para o obter os resultados esperados pelo paradigma SSVEP.

A base de dados utilizada possui duas sessões que utilizam um diferente número de estímulos para obter e gravar os dados presentes, porém os estímulos foram utilizados em diferentes proporções, além de possuir estímulos que apenas aparecem em uma das sessões do experimento. Para exemplificar a proporção desigual encontrada na base de dados, a sessão 2 apresenta metade dos dados utilizando o estímulo 6 Hz (equivalente a 5 vezes mais do que os demais estímulos). A presença de um desbalanceamento tão acentuado nos dados pode causar com que o classificador generalize os dados de forma a pouco representar a realidade. Por conta do desbalanceamento apresentado na segunda sessão do experimento, foi escolhido treinar o classificador utilizando a primeira sessão que não apresenta uma desbalanceamento tão acentuado.

A presença de artefatos também se torna bastante comum em grande parte dos dados, apresentando diversos casos que desviam do esperado dentro do paradigma SSVEP, no qual o pico do sinal no domínio da frequência deveria corresponder ao sinal evocado no experimento. A Figura 6.5 apresenta um comparativo entre um sinal esperado (a esquerda) e o sinal com artefato (a direita).

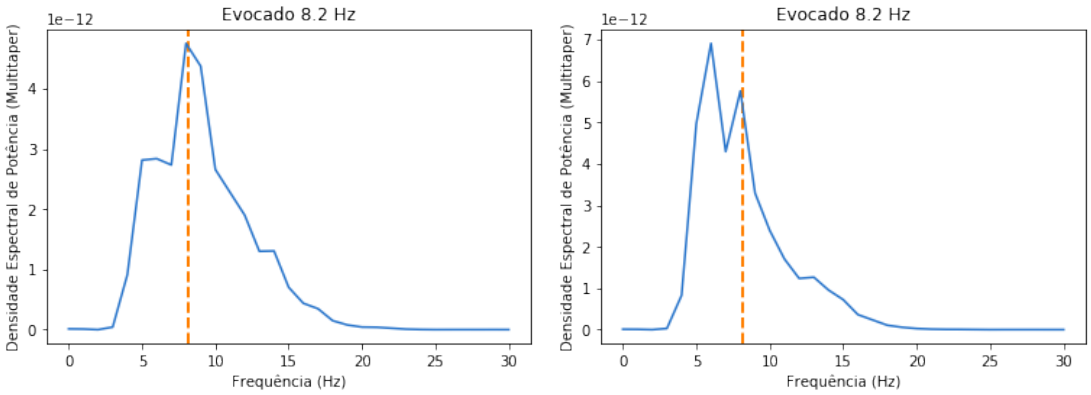

Figura 6.5: Esquerda: Sinal sem a presença de artefatos; Direita: Dados com a presença de artefatos.

Não é possível afirmar qual a causa dos artefatos encontrados, visto que estes podem ser derivados de fatores internos do indivíduo, ou externos da aparelhagem utilizada para gravar o experimento. Na base de dados, todos os indivíduos possuem certos artefatos, interferindo no processo de aprendizagem do classificador. Os indivíduos 1 e 3 foram os que apresentaram a maior quantidade de artefatos.

6.2.3 Resultados do experimento

Devido ao fato de cada indivíduo se comportar de maneira diferente ao realizar o experimento, é necessário analisar os resultados de maneira individual. A Tabela 6.1 apresenta a acurácia obtida pelo classificador para cada indivíduo.

Tabela 6.1: Resultados obtidos pelo classificador para cada um dos 5 indivíduos.

	Indivíduo 1	Indivíduo 2	Indivíduo 3	Indivíduo 4	Indivíduo 5
Resultado	0.5	0.75	0.625	0.75	1

Durante o experimento, os indivíduos 1 e 3 apresentaram os piores resultados comparado com os restantes. Isto ocorre devido a grande presença de artefatos presentes nos dados. Os artefatos encontrados resultaram em picos em frequências diferentes das evocadas no experimento, desviando do esperado pelo paradigma SSVEP. Ao analisar os indivíduos nos quais o sinal respeita o paradigma estudado, foi possível observar uma alta taxa de acerto do classificador.

É importante notar que devido a pequena quantidade dos dados presentes para a sessão de treino e teste, a porcentagem de acerto acaba sendo afetada, visto que o classificador foi treinado apenas com 10 amostras e 8 diferentes amostras foram utilizadas para teste. O pequeno número de amostras é um fator limitante em experimentos BCI, muitas vezes inviabilizando a utilização de algoritmos de aprendizado de máquina mais eficientes.

6.3 Experimento de predisposição para alcoolismo

O experimento realizado utiliza versão grande da base de dados EEG Database Data Set [35], que é dividida entre dois conjuntos de mesmo tamanho, sendo um para treino e outro para teste. Os dois conjuntos possuem a mesma quantidade de experimentos e a divisão entre indivíduos com e sem predisposição para apresentar alcoolismo é de 50% tanto no conjunto de treino quanto o de teste.

Como nos demais experimentos que utilizam uma etapa de classificação, o experimento foi divido em dois, sendo um destinado ao treinamento do classificador e outro destinado a utilização do mesmo para realizar a predição dos resultados.

O experimento apresenta passos distintos dos demais devido a presença de filtros espaciais durante a etapa de pré-processamento. Primeiramente é utilizado um filtro para remover por completo dados de alguns eléctrodos que não representam dados úteis para o experimento, sendo estes denominados de `x`, `nd` e `y`. O outro filtro espacial utilizado foi o algoritmo `car`, utilizado para remover dados de determinados elétrodos dos restantes, os eléctrodos escolhidos após um processo de analise empírica foram os `c3`, `cp4` e `cp5`.

Após aplicar o algoritmo `car` pode ser facilmente notado uma forte distinção entre o PSD de indivíduos com e sem predisposição para o alcoolismo. A Figura 6.6 apresenta a comparação do PSD de indivíduos com e sem predisposição para o alcoolismo.

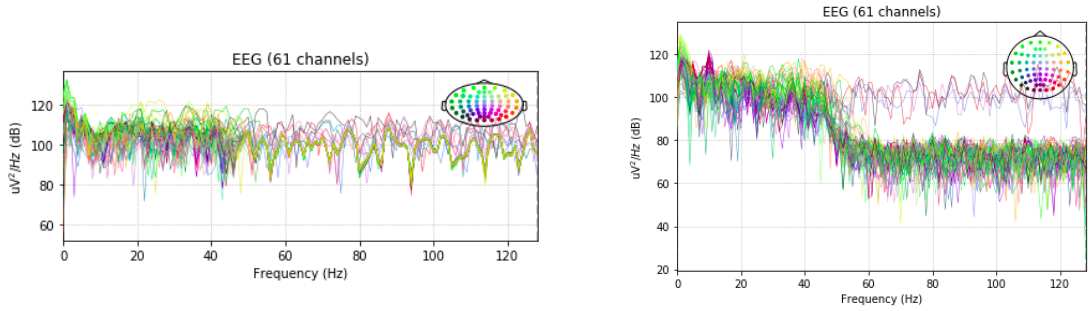

Figura 6.6: Esquerda: PSD de um indivíduo com predisposição a alcoolismo; Direita: PSD de um individuo de controle.

Após todo o processamento, bastou aplicar um simples classificador para ser capaz

de diferenciar os dois tipos de indivíduos. A Figura 6.7 apresenta o fluxo completo utilizado para a etapa de treinamento, enquanto a Figura 6.8 apresenta o fluxo na etapa de classificação.

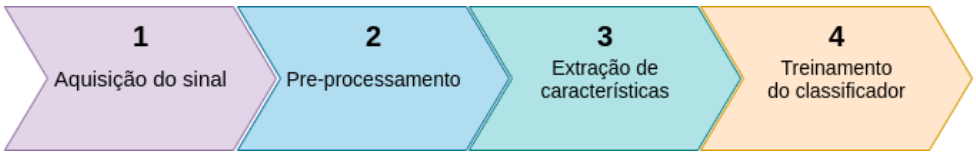

Figura 6.7: Passos necessários para realizar a etapa de treinamento do classificador.

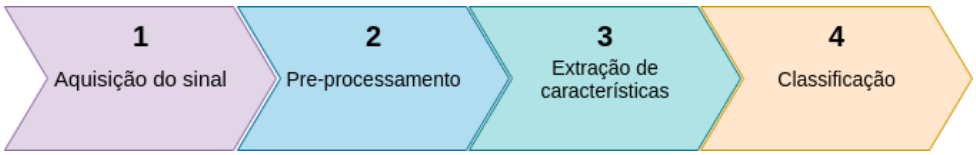

Figura 6.8: Passos necessários para realizar a etapa de classificação do experimento.

Foi utilizado um classificador SVM devido a forte diferença entre as classes e foi obtido uma precisão de aproximadamente **97%**, mostrando a efetividade de um classificador simples para diversos tipos de experimentos BCI. A Listagem 6.3 apresenta de forma resumida a fase de treinamento do modelo de classificação utilizada neste experimento.

Listagem 6.3: Experimento desenvolvido para a etapa de treinamento.

```
...

USELESS_CHANNELS = ['x', 'nd', 'y']
USEFULL_CHANNELS = ["c3", "cp4", "cp5"]

data, labels = getdata_label(
    "Custom", get_data=test_data.get_data, get_label=test_data.get_label)
objs = create_eeg_object(data, 256, channels=test_data.ch_names)
dropped_channels_objs = drop_channels(objs, USELESS_CHANNELS)
car_objs = car(dropped_channels_objs, USEFULL_CHANNELS)
features = extract(car_objs, [PSD()])
classifier = training('SVM', features, labels, verbose=True)
```

```
classifier.save('model.joblib')
```

6.4 Considerações finais

Após as realizações dos testes propostos, foi notado como o *toolkit* foi capaz de facilitar grande parte das funcionalidades, tornando as etapas comuns a todos os experimentos simples de serem realizadas, enquanto permite a utilização de funções mais especificas em suas etapas, garantindo que casos mais específicos possam ser contemplados pelo *toolkit*.

Capítulo 7

Conclusões e trabalhos futuros

Nesta tese foi desenvolvido um *toolkit* com o objetivo de facilitar e unificar as ferramentas necessárias para o desenvolvimento de um experimento BCI. Foi analisado 3 diferentes tipos de experimentos com características distintas, avaliando a capacidade e eficácia do *toolkit* desenvolvido para auxiliar na realização dos experimentos. Como resultado do desenvolvimento desta tese foi gerado um produto com um potencial de facilitar e auxiliar novas pesquisas dentro do escopo de BCI.

Foi concluído que existe uma viabilidade de desenvolver uma solução unificada para auxiliar novas pesquisas, apresentando uma única solução a ser estudada para que novos pesquisadores possam utilizar, concluindo assim o objetivo principal do trabalho desenvolvido.

Devido a característica do trabalho, o desenvolvimento foi focado nos módulos necessários para os experimentos realizados, tornando os demais módulos e integrações como possíveis trabalhos futuros. Alguns dos possíveis trabalhos futuros estão listados a seguir.

- Explorar diferentes tipos de experimentos e paradigmas diferentes dos que já foram utilizados, durante o desenvolvimento deste trabalho, foi focado principalmente em paradigmas visuais, sendo um possível trabalho futuro a adição dos demais paradigmas nativamente no *toolkit*.

- Durante o desenvolvimento do projeto, foi implementado métodos de processamento

que foram necessários para o escopo do experimento, um possível trabalho futuro pode consistir em levantar os principais métodos de processamento do sinal, e implementar no *toolkit* aqueles que ainda não foram contemplados.

- Explorar diferentes fluxos de experimento, com diferentes formas de aquisição do sinal, diferenciando para experimentos *online* e *offline*.

- Expandir a interface gráfica desenvolvida, de modo a contemplar mais funções e melhorar o delay de comunicação com o *toolkit*.

Bibliografia

[1] S. N. d. C. Leite et al., "Contribuições ao desenvolvimento de interfaces cérebro-computador baseadas em potenciais evocados visualmente em regime estacionário", tese de doutoramento, Universidade Estadual de Campinas, 2016.

[2] R. Hübner et al., "Desenvolvimento de um sistema ssvep-bci para o auxílio em tomada de decisões", tese de doutoramento, Universidade Estadual de Campinas, 2018.

[3] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg et al., "Scikit-learn: Machine learning in python", *Journal of machine learning research*, vol. 12, nº Oct, pp. 2825–2830, 2011.

[4] R. A. Ramadan, S. Refat, M. A. Elshahed e R. A. Ali, "Basics of brain computer interface", pp. 31–50, 2015.

[5] M. R. Lakshmi, T. Prasad e D. V. C. Prakash, "Survey on eeg signal processing methods", *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 4, nº 1, 2014.

[6] L. F. Nicolas-Alonso e J. Gomez-Gil, "Brain computer interfaces, a review", *Sensors*, vol. 12, nº 2, pp. 1211–1279, 2012.

[7] C. Kothe, *Lab streaming layer (lsl)*, Acessado em 13/05/2019, 2014. endereço: `https://github.%20com/sccn/labstreaminglayer`.

[8] D. J. McFarland, C. W. Anderson, K.-R. Muller, A. Schlogl e D. J. Krusienski, "Bci meeting 2005-workshop on bci signal processing: Feature extraction and translation", *IEEE transactions on neural systems and rehabilitation engineering*, vol. 14, nº 2, pp. 135–138, 2006.

[9] J. Wolpaw e E. W. Wolpaw, *Brain-computer interfaces: Principles and practice*. 198 Madison Avenue New York, NY 10016 USA: Oxford University Press, 2012, ISBN: 9780195388855.

[10] M. J. Alhaddad, "Common average reference (car) improves p300 speller", *International Journal of Engineering and Technology*, vol. 2, nº 3, p. 21, 2012.

[11] T. Saramaeki, S. Mitra e J. Kaiser, "Finite impulse response filter design", *Handbook for digital signal processing*, vol. 4, pp. 155–277, 1993.

[12] K. A. Ludwig, R. M. Miriani, N. B. Langhals, M. D. Joseph, D. J. Anderson e D. R. Kipke, "Using a common average reference to improve cortical neuron recordings from microelectrode arrays", *Journal of neurophysiology*, vol. 101, nº 3, pp. 1679–1689, 2009.

[13] N. Chumerin, N. V. Manyakov, M. Van Vliet, A. Robben, A. Combaz e M. M. Van Hulle, "Preprocessing and decoding steady-state visual evoked potentials for brain-computer interfaces", *Digital Image and Signal Processing for Measurement Systems*, pp. 1–33, 2012.

[14] N. Chumerin, N. V. Manyakov, M. van Vliet, A. Robben, A. Combaz e M. M. Van Hulle, "Steady-state visual evoked potential-based computer gaming on a consumer-grade eeg device", *IEEE transactions on computational intelligence and ai in games*, vol. 5, nº 2, pp. 100–110, 2012.

[15] P. Welch, "The use of fast fourier transform for the estimation of power spectra: A method based on time averaging over short, modified periodograms", *IEEE Transactions on audio and electroacoustics*, vol. 15, nº 2, pp. 70–73, 1967.

[16] B. E. Boser, I. M. Guyon e V. N. Vapnik, “A training algorithm for optimal margin classifiers”, pp. 144–152, 1992.

[17] C. Vidaurre, A. Schlögl, B. Blankertz, M. Kawanabe e K.-R. Müller, “Unsupervised adaptation of the lda classifier for brain–computer interfaces”, vol. 2008, pp. 122–127, 2008.

[18] Y. Li, J. Pan, F. Wang e Z. Yu, “A hybrid bci system combining p300 and ssvep and its application to wheelchair control”, *IEEE Transactions on Biomedical Engineering*, vol. 60, nº 11, pp. 3156–3166, 2013.

[19] E. Yin, Z. Zhou, J. Jiang, Y. Yu e D. Hu, “A dynamically optimized ssvep brain–computer interface (bci) speller”, *IEEE transactions on biomedical engineering*, vol. 62, nº 6, pp. 1447–1456, 2014.

[20] A. A. Ghodake e S. Shelke, “Brain controlled home automation system”, pp. 1–4, 2016.

[21] B. Venthur, “Design and implementation of a brain computer interface system”, tese de doutoramento, Technische Universität Berlin, Fakultät IV - Elektrotechnik und Informatik, mar. de 2015.

[22] Z. Tayeb, N. Waniek, J. Fedjaev, N. Ghaboosi, L. Rychly, C. Widderich, C. Richter, J. Braun, M. Saveriano, G. Cheng et al., “Gumpy: A python toolbox suitable for hybrid brain–computer interfaces”, *Journal of neural engineering*, vol. 15, nº 6, p. 065 003, 2018.

[23] M. Michalska, “Openbci: Framework for braincomputer interfaces”, tese de doutoramento, University of Warsaw Faculty of Mathematics, Informatics e Mechanics, 2009.

[24] K. C. Louden et al., *Programming languages: Principles and practices*. PWS-Kent (1993): Cengage Learning, 2011.

[25] G. Lindstrom, “Programming with python”, *IT professional*, nº 5, pp. 10–16, 2005.

[26] A. Gramfort, M. Luessi, E. Larson, D. A. Engemann, D. Strohmeier, C. Brodbeck, R. Goj, M. Jas, T. Brooks, L. Parkkonen et al., "Meg and eeg data analysis with mne-python", *Frontiers in neuroscience*, vol. 7, p. 267, 2013.

[27] A. Gramfort, M. Luessi, E. Larson, D. A. Engemann, D. Strohmeier, C. Brodbeck, L. Parkkonen e M. S. Hämäläinen, "Mne software for processing meg and eeg data", *Neuroimage*, vol. 86, pp. 446–460, 2014.

[28] Y. Renard, F. Lotte, G. Gibert, M. Congedo, E. Maby, V. Delannoy, O. Bertrand e A. Lécuyer, "Openvibe: An open-source software platform to design, test, and use brain–computer interfaces in real and virtual environments", *Presence: Teleoperators and virtual environments*, vol. 19, nº 1, pp. 35–53, 2010.

[29] S. Van Der Walt, S. C. Colbert e G. Varoquaux, "The numpy array: A structure for efficient numerical computation", *Computing in Science & Engineering*, vol. 13, nº 2, p. 22, 2011.

[30] P. Barrett, J. Hunter, J. T. Miller, J.-C. Hsu e P. Greenfield, "Matplotlib–a portable python plotting package", vol. 347, p. 91, 2005.

[31] S. Tilkov e S. Vinoski, "Node. js: Using javascript to build high-performance network programs", *IEEE Internet Computing*, vol. 14, nº 6, pp. 80–83, 2010.

[32] E. Gamma, R. Helm, R. Johnson e J. Vlissides, *Design patterns: Elements of reusable object-oriented software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995, ISBN: 0-201-63361-2.

[33] I. Fette e A. Melnikov, "The websocket protocol", 2011. DOI: 10.17487/RFC6455. endereço: https://www.rfc-editor.org/rfc/rfc6455.txt.

[34] A. Vilic, *Avi ssvep dataset*, http://www.setzner.com/avi-ssvep-dataset/, Acessado em 19/05/2019, 2014.

[35] H. Begleiter, "Eeg database data set", 1995, Acessado em 19/05/2019.

[36] P. R. De Andrade, A. B. Albuquerque, O. F. Frota, R. V. Silveira e F. A. da Silva, "Cross platform app: A comparative study", *ArXiv preprint arXiv:1503.03511*, 2015.