Lucas Kenji Kawamura
Rafael Pereira Piza Palmério

Desenvolvimento de plataforma para controle de dispositivo externo em tempo real por meio de sinais eletrofisiológicos

São Paulo
2017

Lucas Kenji Kawamura
Rafael Pereira Piza Palmério

Desenvolvimento de plataforma para controle de dispositivo externo em tempo real por meio de sinais eletrofisiológicos

Trabalho de formatura submetido à Escola Politécnica da Universidade de São Paulo como requisito para a obtenção de diploma de graduação em Engenharia Mecatrônica

Universidade de São Paulo - USP
Escola Politécnica
Departamento de Engenharia Mecatrônica

Orientador: Prof. Dr. Arturo Forner-Cordero

São Paulo
2017

Lucas Kenji Kawamura
Rafael Pereira Piza Palmério

Desenvolvimento de plataforma para controle de dispositivo externo em tempo real por meio de sinais eletrofisiológicos/ Lucas Kenji Kawamura e Rafael Pereira Piza Palmério. - São Paulo, 2017.

Orientador: Prof. Dr. Arturo Forner-Cordero

Trabalho de formatura - Universidade de São Paulo - USP

Escola Politécnica

Departamento de Engenharia Mecatrônica, 2017.

CDU xx:xxx:xxx.x

AGRADECIMENTOS

Agradecemos aos membros do Laboratório de Biomecatrônica do Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos, que acompanharam o desenvolvimento do projeto desde o início, prestando auxílio e fornecendo recursos materiais e intelectuais. Em especial, agradecemos ao prof. Dr. Arturo Forner-Cordero e ao prof. Rafael Traldi Moura, orientador e co-orientador deste trabalho, respectivamente, pelas críticas e sugestões prestadas durante a evolução do projeto.

Prestamos também agradecimentos aos nossos pais, familiares e amigos, que nos apoiaram e incentivaram em todos os momentos.

RESUMO

Este documento descreve o projeto de uma Interface Cérebro-Máquina baseada na identificação de sinais SSVEP para controle de dispositivos externos, cobrindo tanto as etapas de desenvolvimento como de validação do sistema. O projeto foi realizado com o apoio do Laboratório de Biomecatrônica da Escola Politécnica da Universidade de São Paulo e poderá ser integrado a outros sistemas em desenvolvimento dentro do laboratório. A etapa de desenvolvimento descreve os equipamentos utilizados para aquisição dos sinais EEG e para a excitação dos sinais eletrofisiológicos, os procedimentos realizados para o tratamento dos dados adquiridos e, por fim, o algoritmo de classificação adotado. Os experimentos realizados inicialmente resultaram em uma performance ruim do sistema, sendo levantadas algumas hipóteses e realizados testes que levaram a concluir que o mau funcionamento da plataforma de aquisição foi possivelmente a causa do problema. Embora não tenham sido realizados testes de integração com outros dispositivos, pode-se presumir que a BCI projetada poderia potencialmente atingir os objetivos propostos.

Palavras-chave: BCI, Eletroencefalografia, Biomecatrônica

ABSTRACT

This document describes the project of a Brain Machine Interface based on the identification of SSVEP signals in order to control external devices, presenting the development and system validation stages. The project was accomplished with the support of the Biomechatronics Laboratory of the Escola Politécnica da Universidade de São Paulo and will be possibly integrated with other systems under development in the Laboratory. The development stage describes the equipments used for acquiring the EEG signals and for exciting the electrophysiological signals, the procedures used for processing the acquired data and, finally, the selected classification algorithm. The experiments initially resulted in a bad performance of the system, and some hypothesis were raised and tests were performed leading to the conclusion that the malfunctioning of the acquisition platform was probably the cause of the problem. Although integration tests with external devices were not performed, it is possible to presume that the presented BCI could potentially achieve the proposed objectives.

Keywords: BCI, Electroencephalography, Biomechatronics

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

BCI	Interface Cérebro-Computador (do inglês *Brain-Computer Interface)*
BLC	Classificador Linear de Bayes (do inglês *Bayes Linear Classifier*)
BMI	Interface Cérebro-Máquina (do inglês *Brain Machine Interface)*
CAR	Referência de Média Comum (do inglês *Common Average Reference*)
CSP	Padrão Espacial Comum (do inglês *Common Spatial Pattern*)
EEG	Eletroencefalograma (do inglês *Electroencephalography)*
EMG	Eletromiografia (do inglês *Electromyography)*
EOG	Eletro-oculografia (do inglês *Electrooculography)*
FFT	Transformada rápida de Fourier (do inglês *Fast Fourier Transform)*
ITR	Taxa de Transferência de Informação(do inglês *Information Transfer Rate*)
LDA	Análise por Discriminante Linear (do inglês *Linear Discriminant Analysis*)
SNR	Razão Sinal-Ruído (do inglês *Signal to Noise Ratio)*
ANN	Redes Neurais Artificiais (do inglês *Artificial Neural Network*)
SSVEP	Potenciais Visuais Evocados de Estado Estacionário(do inglês *Steady-State Visual Evoked Potentials)*
SVM	Máquina de Vetores de Suporte (do inglês *Support Vector Machine*)

SUMÁRIO

1. INTRODUÇÃO

A biomecatrônica surgiu da interação entre a biomecânica, o estudo da mecânica dos seres vivos, e da mecatrônica, área de integração de conhecimentos de mecânica e eletrônica, particularmente da robótica. Essa ciência integra conhecimentos da área de engenharia com áreas como medicina e fisioterapia. Para isso, são estudados movimentos e sinais eletrofisiológicos do corpo humano e desenvolvidos sistemas robóticos para interagir com estes elementos.

A interação de um ser humano com um dispositivo externo deve ser feita primeiramente por meio de sensores, que podem captar a trajetória, velocidade e acelerações de pontos ao longo do corpo, ou então, por meio de eletrodos, realizar a leitura dos potenciais elétricos produzidos pelo cérebro e transmitidos aos restante do corpo.

Um sinal captado por um sensor pode então ser interpretado por um sistema e ser empregado em atuadores para produzir sons ou movimentos, por exemplo. Por fim, esses atuadores podem agir em conjunto e controlar um dispositivo externo, como uma prótese robótica (máquina mecatrônica que atua no lugar de um membro perdido) ou um exoesqueleto robótico (máquina mecatrônica que atua em paralelo a um membro).

O presente trabalho foca no desenvolvimento de uma Interface Cérebro-Máquina (em inglês, *Brain-Computer Interface* ou *BCI*), sistema que integra sinais eletrofisiológicos de um ser humano com respostas em tempo real de um dispositivo externo.

2. OBJETIVOS

Este trabalho tem por objetivo o desenvolvimento de uma interface cérebro-máquina capaz de captar sinais eletrofisiológicos via métodos não-invasivos e processá-los de forma a controlar um dispositivo externo. O sistema poderá ser integrado com outros sistemas desenvolvidos no laboratório de biomecatrônica, como por exemplo o braço de um exoesqueleto (projetos de TCC e pós-graduação), um ambiente de realidade virtual (projeto de TCC) e uma montagem experimental para estudo de timing coincidente (projeto de TCC). A interface cérebro-máquina deve atuar sobre os dispositivos externos de maneira a gerar 2 variáveis de controle, como por exemplo a velocidade angular e o torque, para cada grau de liberdade.

O projeto abrange as áreas de software e processamento de sinais sendo as principais atividades a serem desenvolvidas, brevemente descritas abaixo.

Para a etapa de pré-processamento dos sinais serão estudados e implementados os algoritmos para o tratamento dos sinais de EEG, permitindo a obtenção de sinais com qualidade adequada à realização do processamento. Serão utilizados no projeto métodos para a obtenção e diferenciação das características pertinentes dos estados mentais do usuário, processo chamado de extração de atributos (em inglês, *feature extraction*). Os atributos extraídos do sinal devem então passar por um algoritmo de classificação que interprete o estado mental do usuário e envie um sinal de comando para o sistema externo, que deve então gerar uma saída correspondente.

Finalmente, serão feitas experiência de validação cujo objetivo é comprovar o funcionamento do ambiente construído durante o projeto.

3. REVISÃO BIBLIOGRÁFICA

3.1. Procedimento de Busca

As pesquisas foram realizadas por meio do portal de Busca Integrada da USP, sendo os artigos obtidos nas bases da PubMed, IEEE e Scopus utilizando as seguintes palavras-chave: BCI, Brain Machine Interface, Human Machine Interface, Real Time, Wheelchair, MATLAB, embedded system, EEG, EMG, EOG.

As palavras chave foram escolhidas de acordo com algumas das especificações do projeto como, por exemplo, execução em tempo real, processamento em um sistema embarcado, utilização de EEG. Ao longo da pesquisa, surgiram referências a outros trabalhos e termos recorrentes entre eles, como controle de cadeira de rodas via BCI, aquisição de sinais via EMG ou interfaces em ambiente Matlab. Assim, artigos frequentemente mencionados nos demais receberam prioridade durante a seleção de novos textos.

O procedimento de busca iniciou-se com a pesquisa do termo *Brain Computer Interface*, porém este termo gerou um número excessivo de resultados (13.323 resultados no Scopus e 5510 no IEEE). Para refinar a pesquisa, utilizamos termos relacionados aos requisitos de projeto, como *Real Time* (3.959 no Scopus e 757 no IEEE) e, em seguida, EEG, o que reduziu o número de resultados (2.723 no Scopus e 360 no IEEE). Assim, iniciou-se a leitura a partir dos artigos classificados como mais relevantes. Numa segunda etapa, passamos a utilizar alguns termos mais específicos encontrados na leitura e que poderiam servir de palavras-chave, como electric wheelchair e 2 degrees of freedom robot.

Foram utilizados para a revisão bibliográfica um total de 20 artigos. Há uma grande quantidade de publicações relacionadas ao tema, porém foram selecionadas as mais pertinentes ao assunto e que apresentavam com maior detalhes as tecnologias empregadas para a implementação do sistema. Dessa maneira, esta revisão bibliográfica pode ainda ser considerada parcial e, provavelmente, expandida no futuro.

3.2. Aquisição e pré-processamento de sinais

Diversas pesquisas ao redor do mundo têm aplicado a interface cérebro-máquina para diferentes finalidades, desde a previsão de movimentos oculares em macacos até o controle de cadeiras de rodas elétricas. A tecnologia das BCIs incorpora diferentes subsistemas, dentre eles estão os circuitos de amplificação e tratamento dos sinais, as plataformas para processamento em Real-Time e tecnologias de aquisição dos sinais eletrofisiológicos (EEG, EMG, EOG).

A maioria dos trabalhos revisados fazem uso somente da EEG para a aquisição dos sinais, mas existe a possibilidade de utilizá-la em conjunto com outras tecnologias, como EMG (EL-MADANI et al., 2012; FERREIRA et al., 2008) e EOG (HORTAL, IÁÑEZ et al., 2015). Também existem pesquisas em que a abordagem envolve o uso de sensores que captam outros tipos de estímulos, como câmeras para o reconhecimento da posição dos olhos (TAKAHASHI, KASHIYAMA, 2005), ou ainda a percepção do ambiente por meio de sensores de distância, auxiliando no controle de uma cadeira de rodas via EEG (GALÁN et al., 2008).

Um fator que varia entre as pesquisas é a escolha dos estados mentais utilizados no controle do dispositivo. Muitos utilizam a imaginação da movimentação das mãos como um estado para execução de uma ação (BUENO et al., 2013; HORTAL, PLANELLES et al., 2015; KIM et al., 2013; FERREIRA et al., 2008). Essa escolha é importante, pois interfere diretamente na dificuldade de obtenção de um sinal com qualidade.

Algumas pesquisas (GUNEYSU, AKIN, 2013; PARINI et al., 2009) utilizam como padrões mentais os sinais SSVEP (Steady State Visual Evoked Potentials), que são respostas à estímulos visuais piscantes a uma frequência constante. As frequências de excitação geram sinais observáveis nos neurônios do córtex visual que são traduzidos como picos no espectro de energia do sinal na mesma frequência de excitação e em seus harmônicos. BCIs baseados neste tipo de paradigma são controlados com o usuário exposto a fontes luminosas piscando a diferentes frequências, sendo cada uma

delas associada a uma escolha. Assim, a classificação das features obtidas pode ser realizada identificando-se as maiores amplitudes na FFT dos sinais adquiridos via EEG, e que devem corresponder à frequência ao qual o usuário está olhando diretamente.

Embora os sinais SSVEP podem ser induzidos em uma gama de frequências que varia de 1 Hz até 100 Hz, pesquisas apontam que a intensidade dos sinais gerados em baixas frequências (6 Hz a 18 Hz) são mais elevadas, o que os torna mais indicados para aplicações em Interfaces cérebro-máquina.

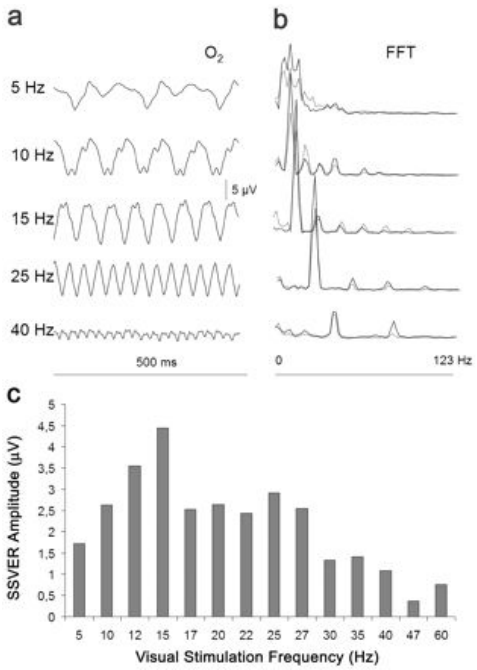


Fig.1 - Effect of visual stimulation at different frequencies on the amplitude of the SSVER. a, SSVER recordings of one representative subject at the right occipital lead (O_2). b, FFT of this individual's SSVERs and grand average of 16 normal subjects (dotted line). c, Average of the mean values of the amplitude of the FFT fundamental frequency of the SSVER recorded at the three occipital leads (Oz, center; O1, left; O_2, right) at the different stimulation frequencies. The amplitude of the occipital SSVER, expressed in microvolts, reached a maximum at ~15 Hz and then fell with a plateau up to 27 Hz, declining at higher frequencies. (PASTOR et al. 2003).

Em praticamente todos os trabalhos analisados, realiza-se o tratamento dos sinais adquiridos antes de serem processados pelos algoritmos de *feature extraction.* Os sinais são aquisitados à frequências de no mínimo 128 hz, são amplificados e logo em seguida filtrados, inicialmente por filtros passa-banda, de modo a eliminar ruídos de alta frequência (usualmente acima 50 hz para EEG). Posteriormente pelo método de *windowing,* reduzindo a influência das componentes de alta frequência que surgem da FFT do sinal truncado.

3.3. Processamento

Depois de obtido um sinal mais limpo, segue-se para a etapa de processamento, na qual são selecionadas características específicas do sinal, para que se possa diferenciar as diferentes áreas ativas do cérebro. Normalmente, as *features* são uma determinada quantidade de amplitudes e frequências dos sinais, obtidas por meio de uma FFT. Em alguns trabalhos, dados como média logarítmica e absoluta também são utilizados para diferenciação entre os padrões mentais.

Após a extração das features, estas são comparadas com um modelo do padrão mental previamente construído *offline*. A diferenciação é feita com uso de algoritmos como o Bayes Linear Classifier (BLC) (EL-MADANI et al., 2015), Linear Discriminant Analysis (LDA) (AFDIDEH et al., 2012; KIM et al., 2013), Support Vector Machine (SVM) (HORTAL, PLANELLES et al., 2015; HORTAL, IÁÑEZ et al., 2015), dentre outros. A utilização de algoritmos de Machine Learning como os acima citados permite a adaptação do classificador a cada usuário, além de possibilitar, dependendo do tipo de implementação, a melhora da acurácia das previsões ao longo do tempo.

O software para processamento dos dados em tempo real é, na maioria das vezes, o MATLAB/Simulink (BUENO et al., 2013; HORTAL, IÁÑEZ et al., 2015; HORTAL, ÚBEDA et al., 2014; AFDIDEH et al.,2012; HORTAL, PLANELLES et al., 2015; KIM et al., 2013). Trata-se de uma ferramenta bastante difundida para o processamento dos sinais. Na maioria dos casos os dados são enviados a um PC ou sistema embarcado para realizar a extração e classificação das *features*.

3.4. Aplicações e Resultados

Por fim, os dados de saída podem ser utilizados tanto para mover um dispositivo físico, como um braço robótico ou uma cadeira de rodas, até para deslocamento dentro de um ambiente de realidade virtual.

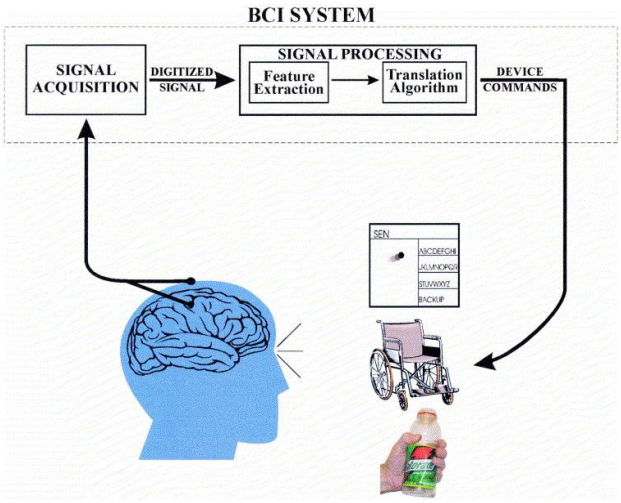


Fig.2 - Basic design and operation of any BCI system. Signals from the brain are acquired by electrodes on the scalp or in the head and processed to extract specific signal features (e.g. amplitudes of evoked potentials or sensorimotor cortex rhythms, firing rates of cortical neurons) that reflect the user's intent. These features are translated into commands that operate a device (e.g. a simple word processing program, a wheelchair, or a neuroprosthesis). Success depends on the interaction of two adaptive controllers, user and system. The user must develop and maintain good correlation between his or her intent and the signal features employed by the BCI; and the BCI must select and extract features that the user can control and must translate those features into device commands correctly and efficiently. (WOLPAW et al., 2002)

Foi desenvolvida uma plataforma embarcada para uma BCI que controla os movimentos dentro de um ambiente de realidade virtual por meio da imaginação dos movimentos das mãos direita e esquerda por (BUENO et al., 2013). O sistema consiste

de três níveis de hardware: o primeiro, é responsável pela amplificação e filtragem dos sinais; o segundo, pelo processamento, executando o tratamento dos sinais por janelamento para, em seguida, realizar a *feature extraction* e a classificação por meio do algoritmo SOM ANN (*Self Organizing Map Artificial Neural Network*); o terceiro, é responsável pela comunicação e armazenamento, podendo comunicar-se com um computador via USB e armazenar as informações processadas em um cartão SD. Na área de software, foram utilizados dois programas em Windows, para visualização dos dados obtidos e para a conversão do arquivo salvo no cartão de memória para o uso em MATLAB. O sistema opera em tempo real e o script em MATLAB gera as máscaras que são aplicadas para a classificação dos sinais em tempo real.

Os pesquisadores (GRAF et al., 2014) fizeram uma interface capaz de prever em tempo real o movimento ocular de macacos com base nos sinais neurológicos enviados por pequenos grupos de neurônios no córtex lateral intraparietal. Para isso foram feitos testes de memória onde luzes em um tela acendiam em uma mesma ordem diversas vezes, e a atividade neural para cada movimento ocular foi gravada.Para a identificação dos estados mentais de cada direção de movimentação foi usado um método estatístico chamado inferência de Bayes. Esses dados foram utilizados em uma segunda etapa onde a partir da atividade cerebral do primata, pudesse ser prevista a direção para a qual seu olho se moveria, com uma precisão de até 83%.

O grupo de pesquisadores (HORTAL, IÁÑEZ et al., 2015), da Universidade de Elche, Espanha, criaram uma interface multimodal para o controle dos movimentos em três dimensões de um braço robotico (FANUC LR-Mate 200iB) para tarefas de *Pick and Place*. O sistema combina a aquisição de sinais via EEG e EOG, traduzidas em movimentos no eixo vertical e no plano horizontal, respectivamente. A imaginação do movimento de mãos ocasiona a elevação da garra, enquanto a recitação do alfabeto leva ao movimento no sentido oposto. O ato de piscar se traduz na abertura e fechamento da garra e a movimentação dos olhos para a direita, esquerda, para cima e para baixo, controlam a movimentação no plano horizontal. A classificação das *features*

dos sinais cerebrais é realizada por um *Support Vector Machine Classifier* juntamente com um método estatístico, que leva em conta a acurácia das saídas anteriores. Os sinais da EOG são processados por meio de um filtro de média móvel e um derivativo, que detecta as variações bruscas de sinal. Os resultados demonstram que a maior dificuldade de operação é o controle pela interface cérebro-máquina e que o refinamento do método de classificação poderia reduzir significativamente o tempo necessário para completar a operação.

Em (HORTAL, ÚBEDA et al., 2014), foi desenvolvida uma BCI cujo objetivo era guiar um robô entre dois pontos, em um movimento sobre um plano. Dois métodos de controle desse robô, o *PupArm*, foram feitos: em um deles, o controle hierárquico, o participante do experimento primeiro selecionava a direção do movimento (horizontal ou vertical) e depois o sentido, e no segundo modo de operação, o de controle direcional, girava-se uma seta no sentido horário ou anti-horário que apontava para o sentido que o robô andaria ao final de cada período de 5 segundos. Para esse experimento foram feitos testes com cada participante para selecionar, dentre 12 tarefas possíveis, 2 que produzissem o melhor sinal na saída do EEG, cujo sinal era adquirido a uma frequência de 256 Hz. O sinal neurológico era filtrado com filtros passa-banda, rejeita-faixa, e Laplaciano. Os estados mentais eram identificados pelo método de *Support Vector Machine,* e o software para controle em tempo real do robô utilizado foi o Simulink.

O software MATLAB é amplamente utilizado em aplicações relacionadas à interfaces cérebro-máquina. O trabalho de (AFDIDEH et al., 2012) consistiu no desenvolvimento de uma toolbox em MATLAB para a movimentação em um ambiente de realidade virtual que simula a caminhada pelos cômodos de uma casa. O sistema conta com duas interfaces, a primeira é focada em monitorar os sinais adquiridos via EEG e o desempenho do voluntário em tempo real, enquanto a segunda trabalha especificamente com a renderização do ambiente virtual e apresentação de feedback ao sujeito. Inicialmente, rodam-se testes *offline* para que se treine o classificador LDA (*Linear Discriminant Analysis*), criando um modelo das *features* para as ações de

imaginar o movimento das pernas e das mãos. A extração é feita com base em dados como médias logarítmicas e absolutas, assim como frequências e amplitudes da FFT. Posteriormente, os sinais adquiridos são classificados em tempo real e geram uma saída para o controlador no ambiente virtual.

Em alguns casos, como na pesquisa de (EL-MADANI et al., 2015), a aquisição via EEG é feita paralelamente à aquisição via EMG, esta última tem o intuito de garantir a passividade dos membros durante o experimento e evitar leituras que não sejam causadas somente pela imaginação do movimento. O artigo também apresenta o efeito do aprendizado do uso da BCI. O desempenho do voluntário aumenta em aproximadamente 10% com clara redução do desvio padrão ao longo de três semanas consecutivas.

No trabalho de (TAKAHASHI, KASHIYAMA, 2005) foi desenvolvido um método para monitoração da posição dos olhos de uma pessoa, que em conjunto com os sinais de EEG, foram utilizados para controle de um robô em um movimento planar.

Os pesquisadores (HORTAL, PLANELLES et al., 2015) desenvolveram uma plataforma para controle de um braço robotico Fanuc LR-Mate 200iB em um plano, pelo uso de 2 tarefas mentais e 2 tarefas motoras, e dessa forma mover o braço nas 4 direções possíveis (frente, trás, esquerda, direita). As duas tarefas mentais foram recitar o alfabeto de trás para frente e contar regressivamente de 20 até 0, e as duas tarefas motoras foram o pensar na movimentação de cada uma das mãos (sem realmente executar o movimento). Utilizando EEG com uma frequência de aquisição de 256 Hz, os dados obtidos foram amplificados e filtrados (filtros passa-banda, rejeita-faixa e Laplaciano) e passados ao MATLAB, onde são extraídas as frequências presentes no sinal pelo método de Welch (que utiliza FFT e resulta em um espectro de frequências entre 8 e 36 Hz em intervalos de 2Hz). Para a classificação do sinal em cada estado mental é utilizado o método de *Support Vector Machine*. Se 4 das últimas 5 classificações forem iguais. o movimento correspondente é feito. O resultado foi uma movimentação com precisão entre 60% e 75%, e a movimentação foi feita entres 146 e 170 segundos, tempo bem menor quando comparado a (HORTAL, ÚBEDA et al.,

2014), cujo tempo médio foi 400 e 500 segundos para cada um dos métodos de controle utilizados.

Em (GALÁN et al., 2008), a proposta dos pesquisadores foi a de utilizar uma BCI para o controle de uma cadeira de rodas por meio de comandos obtidos pela EEG. A cadeira de rodas contava com dois sistemas de controle que atuavam em paralelo, o primeiro que era feito pela interface homem-máquina, e o segundo, que era feito por meio do sensoriamento do ambiente ao redor da cadeira de rodas. Os sistemas atuavam de forma a garantir a segurança do cadeirante, fazendo com que, caso o sistema detectasse a falha do indivíduo em controlar o deslocamento da cadeira, o sistema autônomo assumisse o controle dos movimentos. Os testes foram inicialmente rodados em um ambiente de realidade virtual, que simulava os movimentos de uma cadeira de rodas e, posteriormente, tomados em ambiente físico. O experimento comprovou que sistemas assistidos melhoravam a performance dos pacientes que tinham dificuldade de controlar a cadeira de rodas, no entanto, tinham sua performance comprometida caso o paciente tivesse boa habilidade para controlar o veículo e houvesse interferência do sistema autônomo.

Os pesquisadores (KIM et. al, 2013) também desenvolveram uma interface para controle de uma cadeira de rodas. Eles utilizaram 5 estados mentais para poder controlar o aparelho em 5 direções distintas. Por EEG, capta-se a imaginação do movimento dos pés e de cada uma das mãos (os pés atuam sobre o deslocamento para frente da cadeira de rodas, cada uma das mãos corresponde ao movimento para um dos lados e a combinação dos pés com uma das mãos faz com que o aparelho ande na diagonal desejada). Por se tratarem de estados mentais complexos, que requerem até duas atividades ao mesmo tempo, dois métodos de classificação foram usados: *Common Spacial Patterns* (CSP) e *Linear Discriminant Analysis* (LDA). Os sinais eram antes filtrados usando Common *Average Reference* (CAR) e um filtro passa-banda, e a análise do sinal era feita em tempo real pelo *Simulink.* Finalmente, os sinais eram transformados por um sinal de PWM pelo microcontrolador Atmega128. Foi implementado um controle na cadeira de rodas de modo a fazer a parada e aceleração

bem suaves, e os pesquisadores esperam fazer melhorias na cadeira no futuro.

Outra pesquisa envolvendo o controle de cadeira de rodas via BCI foi realizado por (TANAKA et al., 2005). Nos testes, move-se uma cadeira de rodas em um plano para alcançar uma de duas regiões-objetivo. Para isso, foram definidos dois estados mentais, cada um correspondente à atividades predominantes de um hemisfério do cérebro. Nessa pesquisa foram desenvolvidos tanto um método que classificava o estado mental da pessoa entre os chamados *Right Thinking* and *Left Thinking* como um método que aprendia as características dos dois estados mentais em cada pessoa. O sinal obtido por EEG, era adquirido a 1024 Hz e passava por filtros passa-banda. Durante o período de aquisição a cadeira de rodas permanecia parada e ao fim desse tempo ela se movia de acordo com o pensamento da pessoa. Foi o primeira projeto que utilizou BCIs para movimentação de cadeira de rodas em conjunto com EEG, e conseguiu sucesso no alcance da região-objetivo em 80% dos testes.

No trabalho de (FERREIRA et al.,2008), foram realizadas duas interfaces, uma com sinais obtidos por EEG e outra baseada em sinais obtidos por EMG, relacionados à atividade muscular. Nas duas eles captavam a piscada do olho do participante, porém a de EMG detectava o movimento dos músculos levantadores da pálpebra e a de EEG detectava atividade ou não no córtex visual. O sinal em ambos os casos era filtrado, amplificado (principalmente no caso da EEG, por se tratarem de sinais bem mais fracos), e classificado em cada estado. Durante o experimento, o participante recebia uma tela com várias seções que piscavam uma de cada vez, e correspondiam a um mapeamento de um robô em um plano. Ele deveria piscar os olhos quando a seção à que ele quisesse ir acendesse. Os pesquisadores concluem que esse é um modo barato e eficiente para implementar uma BCI.

Os engenheiros da Escola Politécnica Federal de Lausanne (MILLAN et al., 2004) realizaram uma abordagem diferenciada baseada em um conjunto BCI e máquina de estados para o controle de um robô móvel Khepera. O objetivo do experimento consistia em conduzir o robô por diversas trajetórias ao longo de um corredor que conecta diversas salas. O robô, equipado com 8 sensores infra-vermelhos

era capaz de detectar a proximidade de obstáculos e utilizar essa informação para modificar o comportamento do veículo. Para o experimento, os participantes puderam selecionar os estados mentais com os quais se sentissem mais confortáveis: imaginação do movimento de mãos, subtração, movimento de rotação de um cubo ou concatenação de palavras relacionadas. O desempenho dos voluntários chegou à uma relação média de 75% entre os tempos para execução das trajetórias manualmente e via BCI, provando-se uma alternativa interessante para locomoção em ambientes fechados.

4. DEFINIÇÃO DO PROJETO

4.1. Critérios para definição dos requisitos

Baseando-se nos artigos lidos, foi possível visualizar os aspectos mais importantes a serem considerados para a definição das especificações do projeto. Abaixo são apontados os requisitos definidos para cada um dos segmentos do projeto.

4.2. Requisitos de Projeto

a. **Resposta em tempo real (entre 0,1s e 1s).**

Para que o usuário seja capaz de operar o sistema, é necessário que exista a percepção de causalidade entre o comando enviado ao BCI e a resposta do mesmo. Por esse motivo, o usuário deve receber um feedback em um curto período de tempo.

O tempo de resposta permitido será baseado nos trabalhos publicados por pesquisadores da área de ciências da computação (DABROWSKI, MUNSON, 2011; MILLER,1968),nos quais são definidos o máximo atraso permitido em uma interação entre homem e máquina, de forma a não comprometer a relação de causalidade entre uma entrada dada a um sistema e a saída gerada. Os pesquisadores definem tal valor como um período entre 0,1 e 1s.

b. **Controle de ao menos duas variáveis para um grau de liberdade.**

O presente trabalho servirá de apoio a futuros projetos dentro do mesmo tema no Laboratório de Biomecatrônica, assim como poderá ser integrado com outros sistemas em desenvolvimento no laboratório, como a interface háptica para reabilitação (projeto de TCC), o setup experimental para estudo de Timing Coincidente (projeto de TCC) ou um membro de exoesqueleto (projeto de TCC e IC). Para algumas dessas aplicações é de interesse que um número mínimo de variáveis de controle possam ser atuadas pelo sistema, sendo que os graus

de liberdade variam de acordo com a aplicação. Assim, foi definido que pelo menos duas variáveis possam ser controladas pelo sistema.

c. **Aquisição de sinais eletrofisiológicos.**

O sistema será baseado na aquisição de sinais de EEG para que seja feita a interface entre uma pessoa e um dispositivo externo. O número de canais deverá ser suficiente para a identificação dos atributos do sinal sem, no entanto, elevar o tempo de processamento do sinal, prejudicando o tempo de resposta da interface BCI;

d. **Taxa de transferência de informação suficiente.**

O sistema deve transferir informações numa velocidade capaz de atender o requisito de tempo real estabelecido no **item a**. A seleção do paradigma a ser utilizado para a excitação dos estados mentais, assim como o número de eletrodos utilizados para a aquisição são parâmetros correlacionados com a taxa de transferência máxima permitida para o sistema BCI. Assim, este parâmetro definirá o tamanho dos pacotes de dados analisados por vez;

e. **O sistema não deve provocar dano ou desconforto excessivo ao usuário.**

A aquisição dos sinais será realizada via métodos não invasivos, com a leitura dos impulsos elétricos ocorrendo sobre o couro cabeludo do voluntário por meio de eletrodos. As atividades desempenhadas durante os testes e controle do dispositivo também podem causar desconforto físico ou psicológico ao usuário. Por esse motivo, o grau de incômodo causado ao usuário deverá ser avaliado.

A validação deste requisito poderá ser realizada por meio da utilização de Post-Task Usability Questionnaires como o Nasa-TLX ou SEQ, para avaliação

da dificuldade de uso.

5. IMPLEMENTAÇÃO

5.1. Decisões de Projeto

Durante o desenvolvimento do projeto, foram levantadas e testadas diferentes abordagens para a geração dos estímulos visuais, processamento e classificação dos dados. Algumas metodologias não atenderam às especificações do projeto e foram re-avaliadas, sendo eventualmente substituídas. Abaixo são descritas algumas das abordagens que foram utilizadas e a razão de terem sido adotadas ou descartadas.

5.2. Aquisição de sinais

5.2.1. Seleção da plataforma de aquisição

A seleção da plataforma envolveu a avaliação de critérios de custo do equipamento, taxa de amostragem, que implica no requisito de tempo de resposta do sistema, e interface com plataformas de desenvolvimento, como por exemplo, o MATLAB. A pesquisa de mercado baseou-se na comparação entre equipamentos já disponíveis no Laboratório de Biomecatrônica, entre eles o OpenBCI e o Tmsi, e equipamentos recorrentes na revisão bibliográfica.

Por conta do menor custo, disponibilidade e ampla compatibilidade com outros softwares, dado a política OpenSource de sua fabricante, a plataforma Cyton (OpenBCI 2017) foi selecionada como plataforma de aquisição de sinais e o Headset Ultracortex (OpenBCI 2017) com eletrodos secos (Florida Research Instruments 2017), como dispositivo para contato com o crânio do usuário. A placa conta com taxa de amostragem de 250 Hz, transmissão de dados para o PC via bluetooth, isolando o indivíduo de tensões mais elevadas, além de dispor de uma GUI que permite a visualização da streaming de cada canal, a FFT do sinal e a aplicação de filtros notch e passa-banda.

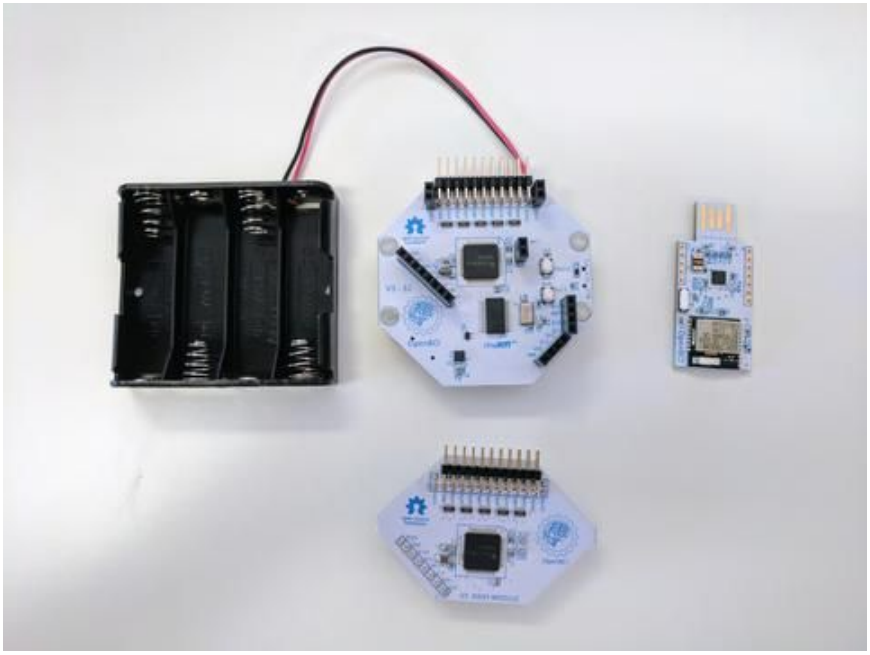


Fig.3 - Plataforma Cyton da fabricante OpenBCI. Extraído de [https://shop.openbci.com/products/cyton-biosensing-board-8-channel?variant=38958638542 em 2/11/2017]

5.2.2 Seleção dos canais a serem utilizados

Foram selecionados 6 canais posicionados de acordo com o sistema 10-20 nas posições O1, O2, P3, Pz, P4, Fz tendo os lóbulos direito e esquerdo (A1 e A2) como BIAS e canal de referência (Fig. 4). Pesquisas apontam que o aumento do número de canais para sinais SSVEP não produz ganho considerável no aumento da taxa de transferência de informação, por tal motivo optou-se por utilizar um número reduzido de eletrodos sobre o lobo occipital, região onde os sinais SSVEP são mais proeminentes, de forma a otimizar o tempo de processamento do sinal.

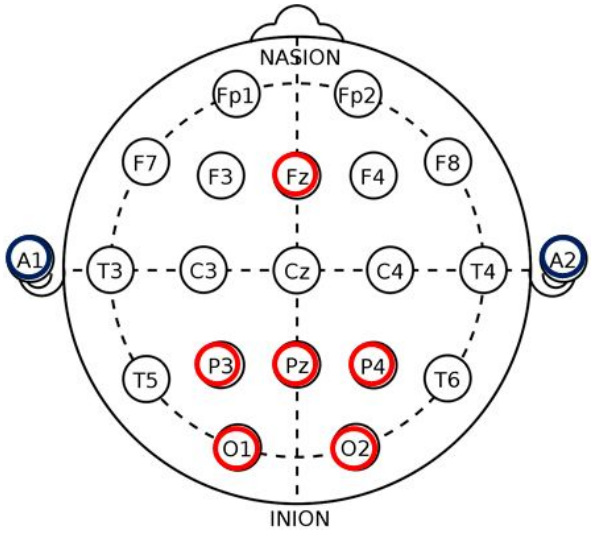


Fig.4 - Posicionamento dos eletrodos de acordo com o sistema internacional 10-20

5.3. Pré-processamento

5.3.1. Definição das etapas de pré-processamento

O processamento dos sinais foi feito offline dividindo-se os 25s de sinal amostrado em epochs de 4s. Para a filtragem do sinal foi aplicado um filtro butterworth de 4ª ordem com frequência de corte de 20 Hz, seguido de janelamento Hanning. Após o tratamento do sinal, segue-se para a etapa de extração dos atributos, que consiste no cálculo da FFT do sinal para obtenção dos valores de amplitudes no espectro de frequências.

Para o pré-processamento dos sinais, também foram testadas outras abordagens como o cálculo das médias entre diferentes canais para eletrodos posicionados em regiões vizinhas, de forma a atenuar ruídos, ou análises para canais individuais, em busca de eletrodos que apresentassem melhor performance.

5.4. Extração de Atributos

5.4.1. Seleção dos atributos a serem identificados

A eletroencefalografia é capaz de detectar uma grande variedade de alterações no campo elétrico do cérebro e, por esse motivo, foram levantados os padrões mentais mais indicados para utilização em interfaces BCI com base na revisão bibliográfica. Alguns dos padrões frequentemente adotados são o Motor Imagery (Kim, Lee,2013), atividades como contagem invertida ou execução mental de operações matemáticas (Hortal et al., 2015) , P300 e o SSVEP (PARINI et al., 2009).

Um exemplo de padrão mental que pode ser facilmente observado na região do lobo occipital, ocorre quando fecham-se os olhos em uma posição relaxada. Enquanto for mantido tal estado, é possível observar uma elevação da amplitude do sinal na faixa em torno dos 10 hz, que corresponde ao valor esperado para as ondas Alfa (Fig. 5).

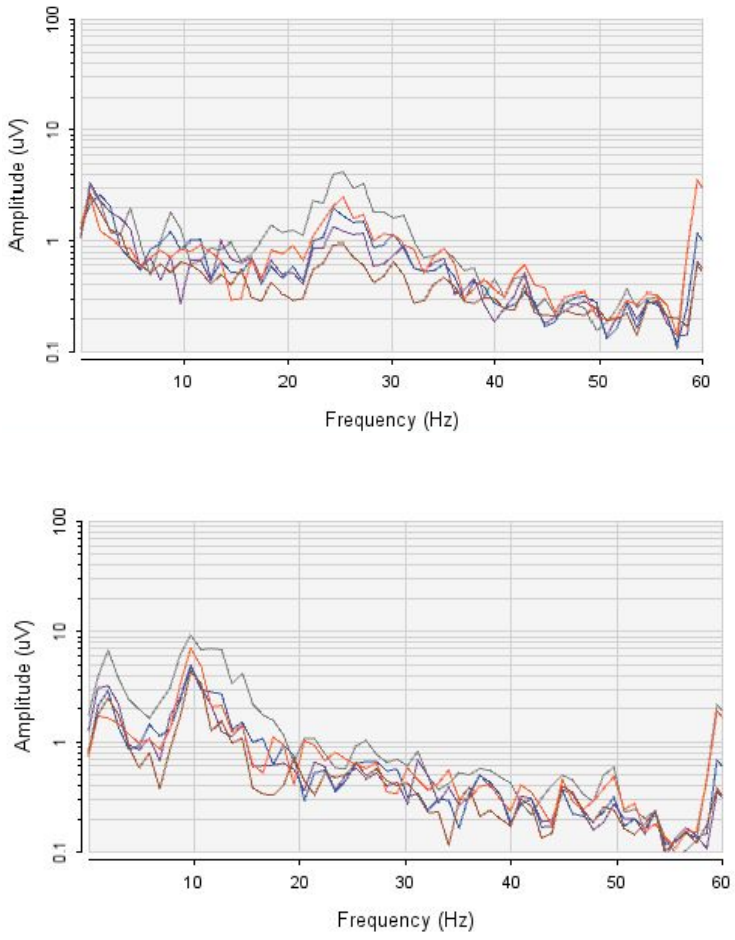


Fig.5 Comparação do espectro de frequências do sinal de E.E.G. com o usuário de olhos abertos (acima) e fechados (abaixo).

Para o presente trabalho foi selecionado o sinal SSVEP como atributo a ser classificado. Os sinais SSVEP, do inglês *Steady-State Visual Evoked Potential*, surge na região do córtex visual como resposta a um estímulo visual repetitivo, como por exemplo, uma lâmpada piscando a determinada frequência em frente a uma pessoa. Este tipo de paradigma é tido como um dos sinais com maior taxa de transferência de informação e menor necessidade de treinamento, além de ser de fácil aprendizagem por parte do usuário, havendo pesquisadores apontando que, com a prática, é possível controlar a intensidade dos sinais SSVEP induzidos no cérebro (MIDDENDORF et al., 2000).

5.4.2 Seleção da interface para excitação dos sinais eletrofisiológicos

Para que fosse possível utilizar o paradigma dos sinais SSVEP como entrada para o sistema BCI, o dispositivo para geração dos estímulos visuais deveria ser capaz de gerar oscilações com frequências bem definidas e estáveis, além de possibilitar a alteração das frequências de excitação de forma simples. A utilização de múltiplas fontes luminosas permite adicionar mais variáveis de controle ao sistema, tornando-o mais versátil e aumentando a variedade de aplicações da BCI.

Foram avaliadas duas abordagens para a geração dos estímulos. A primeira envolve a utilização de uma interface PsychToolbox em MATLAB, que permite a configuração do tamanho e posição dos elementos piscantes, assim como cada uma das frequências associadas ao elemento (Fig.6), enquanto uma segunda alternativa seria a utilização de LEDs, oscilando a diferentes frequências, conectados a um microcontrolador.

A primeira alternativa mostrou-se conveniente do ponto de vista da configuração dos elementos piscantes, dado que o tamanho, posição e frequência de cada fonte luminosa poderia ser facilmente configurada via software. Entretanto, o bom funcionamento deste sistema é dependente da taxa de refresh do monitor e da velocidade de processamento de imagens do computador. Foram realizados alguns

testes com a interface, no entanto, não foi possível visualizar as frequências e harmônicos esperados para um sinal SSVEP.

Fig.6 - Interface apresentada ao usuário durante a execução do script do PsychToolbox. Cada quadrado branco pisca a uma diferente frequência.

Como a segunda abordagem permite maior controle sobre as frequências de excitação, optou-se pela utilização de uma matriz de LEDs 8x8 (circuito integrado MAXIM7219, Maxim Integrated 2017) controlados por uma placa Arduino Due (Arduino 2017), assim como visto em outras pesquisas (GUNEYSU et al., 2013; WU, 2008). Para validar a precisão do método, verificou-se com um osciloscópio os níveis de tensão nas saídas do microcontrolador e constatou-se que as frequências configuradas ficaram bem próximas das frequências obtidas, com erros de no máximo 2%.

5.5. Classificação dos atributos

5.5.1 Seleção dos algoritmos de classificação

Nos trabalhos analisados na revisão bibliográfica a etapa de classificação dos sinais SSVEP é realizado por duas abordagens principais: análise dos picos da Transformada de Fourier e por algoritmos de Machine Learning.

A primeira abordagem consiste na aquisição do sinal cerebral, filtragem e processamento, aplicação da FFT e classificação do estado mental de acordo com a frequência com maior amplitude no espectro de frequências. Em um teste preliminar, a taxa de acertos para frequências de 8hz, 10hz e 12hz foi de aproximadamente 90%. Embora o resultado seja positivo do ponto de vista de interpretação dos sinais, a performance foi baixa dado a necessidade de adquirir um período de amostragem de 25s para a classificação do sinal. Os resultados obtidos em períodos de amostragem inferiores a 10s e maior quantidade de frequências não foram satisfatórios, e mesmo com a variação dos filtros utilizados e de seus parâmetros, a melhor porcentagem de acerto foi de 29%.

A segunda abordagem segue os mesmo princípios básicos da anterior, porém após a aplicação da FFT, o vetor com as amplitudes de cada frequência é utilizado como entrada do classificador baseado no algoritmo de aprendizado de máquina. Foram feitos testes com dois algoritmos diferentes: Redes Neurais Artificiais e Support Vector Machine.

Redes Neurais Artificiais são arquiteturas de aprendizado de máquina supervisionado compostas por camadas de ‘neurônios’. Cada neurônio de cada camada recebe como entrada a saída de cada neurônio da camada anterior (ou da entrada da rede, caso seja a primeira camada). As entradas desse neurônio são multiplicada por um peso (diferente para cada entrada) e somadas. A saída do neurônio é o valor de uma função (por exemplo uma sigmoide) calculada no valor da soma das entradas ponderadas. A saída da rede é sua última camada, e no caso da arquitetura escolhida, o Multilayer Perceptron, o número de neurônios é igual ao número de classes. A classe calculada é a saída de maior valor. Por se tratar de um

algoritmo de aprendizado supervisionado, é necessário, durante o treinamento, fornecer à rede a saída correspondente àquela entrada. Os pesos de cada neurônio são 'aprendidos' por backpropagation, onde a saída e as derivadas de cada peso de cada neurônio são computados para estimar quanto cada peso afeta na decisão da rede, e que, no próximo passo, a saída calculada esteja mais próxima da saída desejada. Isso se repete para cada amostra, várias vezes, até que haja uma convergência. A figura 7 ilustra a arquitetura de uma Rede Neural, onde cada círculo corresponde a um neurônio, cada flecha corresponde a um dos seus pesos, e os quadrados correspondem às entrada da rede.

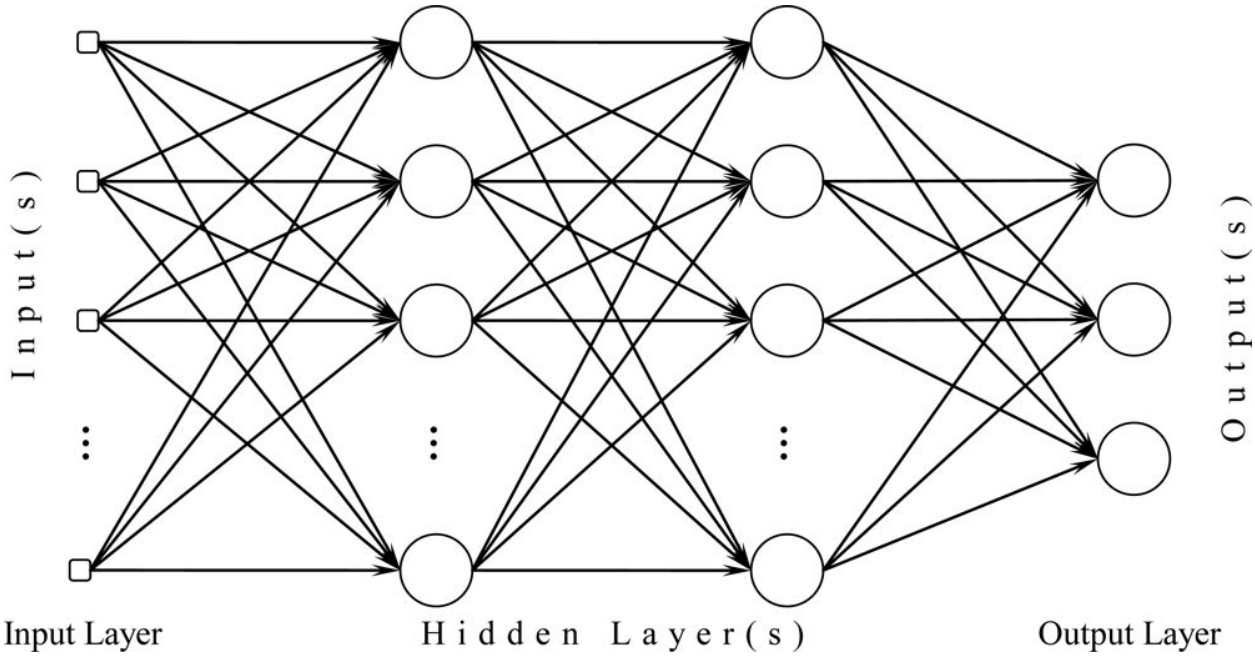


Fig.7 - Arquitetura de uma Rede Neural Artificial. Extraído de http://www.mdpi.com/1424-8220/10/9/8363/htm em 20/10/2017.

Support Vector Machines, conhecidos por SVM, são um algoritmo de aprendizado de máquina supervisionado. Cada amostra e sua respectiva saída são posicionadas num espaço de dimensão n, onde n é o número de variáveis de entrada do algoritmo. O sistema busca encontrar hiperplanos (os planos podem ser lineares ou não-lineares, dependendo do kernel escolhido para os modelos) que melhor separam

as amostras nas suas classes. Em cada passo do algoritmo, é encontrado um conjunto de planos que tenta separar amostras de classes diferentes. Com esse conjunto parcial, calcula-se o valor de uma função de perda, correspondente à quanto o sistema errou com suas classificações. A partir dos vetores entre os pontos e planos, recalculam-se os parâmetros de cada plano de modo a diminuir o valor da função de perda. A figura 8 ilustra a solução de separação de amostras de duas variáveis por um plano, no caso de um kernel não-linear.

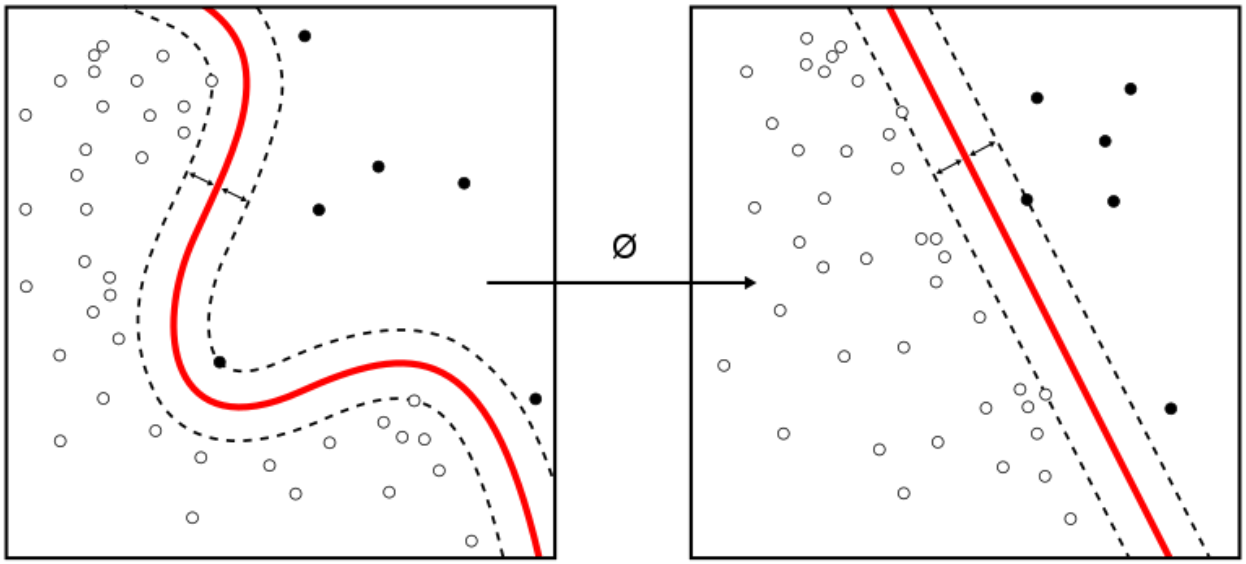


Fig.8 - Separação de amostras pela sua classe, por SVM com kernel não-linear. Extraído de https://en.wikipedia.org/wiki/Support_vector_machine#Computing_the_SVM_classifier em 20/10/2017.

O objetivo das abordagens de algoritmos de aprendizado de máquina é poder tirar conclusões a partir dos dados EEG adquiridos, que não foram facilmente interpretáveis com uma análise de picos da transformada de Fourier, além de conseguir treinar o sistema para cada usuário, podendo abstrair suas características cerebrais individuais. Foram feitos testes de performance com cada algoritmo, e, em média, as redes neurais performaram 2% melhor que o SVM.

5.5.2. Metodologia

Todas as abordagens escolhidas para classificação dos sinais usam diversas outras técnicas já aplicadas para análise dos sinais. Primeiramente, são feitas várias sessões de coleta dos sinais EEG, cada sessão com uma frequência de excitação única. Esses dados são salvos e depois processados. O processamento consiste na agregação de todos os canais, divisão do sinal em epochs, janelamento de cada epoch, filtragem com Butterworth e finalmente é feita a Transformada de Fourier. Em sinais com número de pontos e frequência de aquisição iguais obtém-se o mesmo número de pontos da transformada (importante para rede neural receber os mesmos inputs para todas os experimentos).

Os pontos da transformada de Fourier são as entradas da rede neural. Por se tratar de um treinamento supervisionado, durante o treinamento é necessário dizer à rede qual a saída esperada, e isso é feito pela sessão a que a janela pertence. No experimento feito, foram utilizadas janelas de quatro segundos, que performaram em média 5% acima de outros tamanhos de janela, em cinco frequências: 7, 9, 11, 13 e 15 hz, mais um estado de repouso. As saídas da rede neural, no treinamento e no teste, são respectivamente 0, 1, 2, 3, 4 e 5.

Todas as janelas são utilizadas no treinamento da rede. Para prevenir viés da ordem em que os dados são apresentados ao modelo, é feita uma mistura nas linhas da entrada (garantindo que o número da linha de cada amostra na entrada e saída permaneçam iguais), cada linha é normalizada, e o dataset é dividido aleatoriamente entre treinamento e teste. O modelo finalmente é treinado e testado. A precisão (proporção entre o número de vezes que uma classe foi corretamente prevista e o total de vezes em que aquela classe foi prevista) média obtida no conjunto de dados de teste para todos os experimentos foi de 62%, e o recall (proporção entre o número de vezes que uma classe ocorreu e foi prevista de acordo e o total de vezes que aquela classe ocorreu) médio foi de 60%. Essa taxa de acerto foi melhor que a mesma análise feita somente pelos picos da transformada de Fourier. Caso não houvesse nenhuma

correlação entre as entradas e a saída, esperaria-se uma precisão e recall em torno de 20% (a rede neural tem saídas aleatórias). Esses números, no pior dos casos, poderiam alcançar o valor mínimo de 0%.

5.6. Definição do sistema externo a ser controlado

A última etapa do projeto consiste na integração do sistema BCI com um dispositivo ou sistema externo que responda às saídas produzidas pela interface, ou seja, as classificações dos estados mentais. O dispositivo externo deve devolver algum feedback visual ou sonoro ao usuário, possibilitando o aprendizado e aperfeiçoamento na utilização e controle do dispositivo.

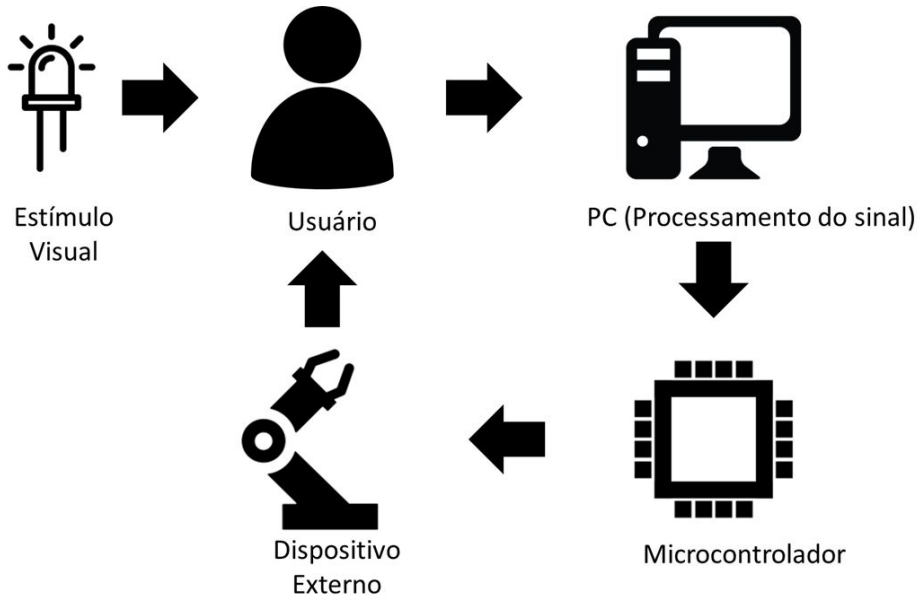


Fig.9 - Arquitetura do Sistema. O estímulo visual acontece por meio de LEDs piscando a diferentes frequências. Os sinais SSVEP que surgem como resposta à excitação são extraídos e classificados no PC. As saídas geradas pelo classificador são enviadas ao microcontrolador que envia os comandos associados a cada uma das frequências a um dispositivo externo.

O sistema a ser controlado consiste de um jogo no qual o usuário tem controle sobre um objeto com formato de espaçonave que pode se mover na direção horizontal e disparar projéteis, totalizando três graus de liberdade para o sistema. O jogo tem

como objetivo atingir os alvos inimigos (coloridos em verde) que se movem também na direção horizontal, aproximando-se da espaçonave a cada vez que atingem uma das fronteiras laterais na tela. Por meio de um marcador de pontuação no canto superior esquerdo, pode-se inferir a performance do usuário no controle do sistema BCI.

Durante a execução do código o classificador transmite a cada segundo um comando que representa a classificação gerada pelo sistema. Como o propósito deste trabalho não é o desenvolvimento de um dispositivo externo que atue com a interface cérebro-máquina, mas que haja a integração com um sistema já existente, o código aberto foi extraído da internet em [http://christianthompson.com/sites/default/files/Space_Invaders/Space-Invaders-Tutorial-10.py], sendo implementada apenas a comunicação entre os ambientes.

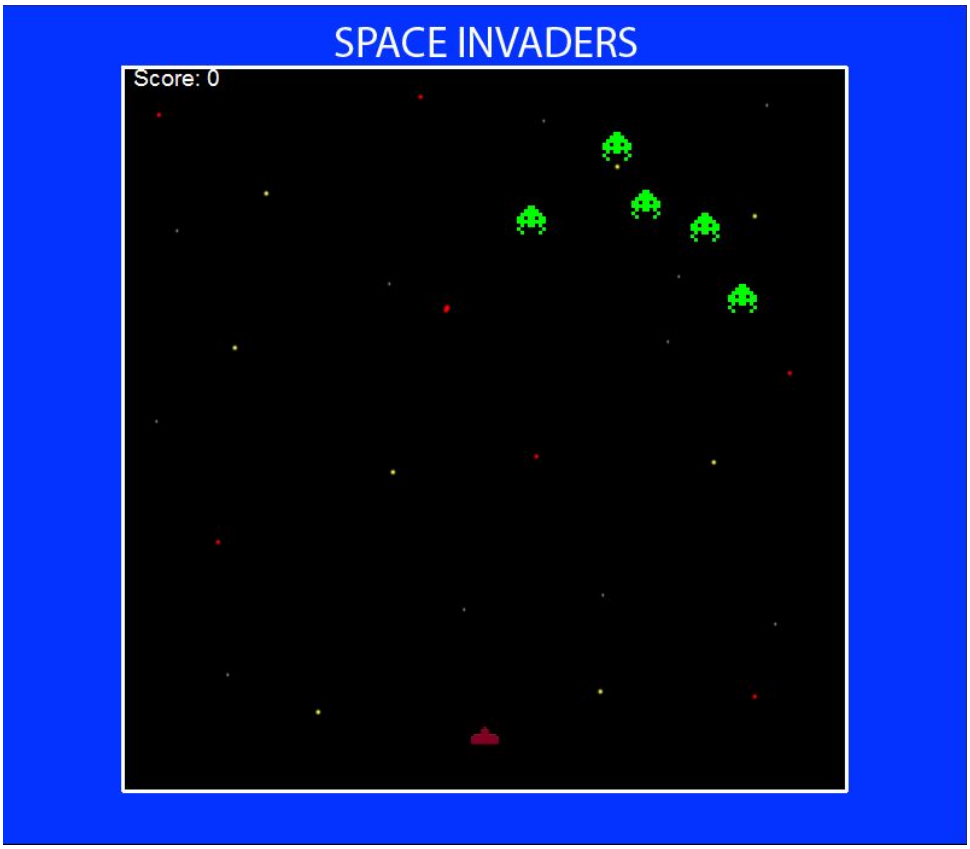


Fig.10 - Tela do jogo tipo "Space Invaders" a ser controlado. Na parte inferior da tela, em vermelho, encontra-se a nave espacial sob controle do usuário. O objeto pode se mover na horizontal e dispara projéteis contra as naves inimigas, destacadas em verde.

5.7. Definição do protocolo experimental

5.7.1. Protocolo de treinamento

O protocolo de treinamento e as frequências a serem utilizadas foram definidas com base nos procedimentos encontrados na revisão bibliográfica, que embora diferentes quanto à natureza dos sinais classificados ou dos dispositivos a serem controlados, seguiam um procedimento experimental similar.

As frequências de excitação utilizadas variaram entre a faixa dos 7hz e 15hz, dado que a intensidade dos sinais SSVEP são mais elevadas para frequências mais baixas (PASTOR et al., 2003). A amostragem foi realizada em trials compostas cada uma de 40s, sendo que os primeiros cinco segundos da amostra eram desconsiderados, dado oscilações geradas no sinal ao iniciar a aquisição. As trials ocorriam em ordens aleatórias, de forma a minimizar a indução de qualquer comportamento padronizado pelo usuário, sendo que cada uma delas era iniciado com uma *cue*, que consistia num comando de voz que indicava o LED a ser observado. Ao final de cada trial era dado um intervalo de 5s ao usuário antes de iniciar a aquisição da próxima frequência.

Fig.11 - Aquisição realizada em trials de 40s, sendo os 5s iniciais não considerados na análise, seguidos de 10s de pausa entre aquisições.

5.7.2 Protocolo de Testes

Após finalizada a etapa de treinamento, inicia-se a fase de testes, na qual o indivíduo é avisado por um comando de voz para qual fonte luminosa deve focar sua atenção. Cada uma das frequências ocorre duas vezes de maneira alternada, com duração de 30s. As saídas geradas pela rede neural vão sendo registradas e comparadas com o valor correto. Ao final do procedimento, os resultados são apresentados em forma de uma matriz de confusão.

6. RESULTADOS E DISCUSSÕES

Os experimentos foram conduzidos com quatro indivíduos diferentes seguindo os procedimentos definidos. As idades dos indivíduos testados eram próximas dos 22 anos, com exceção de um dos usuários, com 59 anos, sendo que dois deles tinham alguma experiência prévia com aquisições de sinais via EEG e EMG. Os experimentos foram conduzidos em salas com baixo nível de luminosidade e o mais distante possível de aparelhos eletrônicos ou fontes de interferência eletromagnética.

Alguns dos usuários obtiveram resultados minimamente satisfatórios durante a etapa de treinamento do classificador, com taxas de acerto próximas aos 70%, como mostrado na Tabela 2. Entretanto, durante a etapa de testes em tempo real a porcentagem de acertos decaiu consideravelmente, o que impossibilitaria o controle de um dispositivo em tempo real. Os dados dos testes são apresentados na Tabela 3.

Informações sobre os usuários				
	Usuário 1	Usuário 2	Usuário 3	Usuário 4
Sexo	F	M	M	M
Idade	20	23	59	22
Experiência com EEG	NÃO	SIM	NÃO	SIM

Tabela 1 - Informações sobre os usuários testados.

Resultado do treinamento realizado para cada usuário				
	Usuário 1	Usuário 2	Usuário 3	Usuário 4
Frequência 1	36%	62%	50%	44%
Frequência 2	30%	36%	64%	73%
Frequência 3	38%	85%	60%	50%
Frequência 4	64%	78%	73%	60%
Frequência 5	40%	58%	82%	57%

Média	42%	67%	68%	57%

Tabela 2 - Resultados do treinamento do classificador para cada usuário.

Resultado dos testes em tempo real para cada usuário				
	Usuário 1	Usuário 2	Usuário 3	Usuário 4
Frequência 1	23%	17%	17%	15%
Frequência 2	26%	56%	13%	14%
Frequência 3	29%	90%	30%	33%
Frequência 4	24%	42%	12%	22%
Frequência 5	21%	74%	27%	86%
Média	24%	58%	20%	36%

Tabela 3 - Resultados do classificador em tempo real

Algumas hipóteses foram levantadas para a possível causa do problema:

- Eletrodos desgastados ou oxidados pelo excesso de uso;
- Fonte luminosa de intensidade insuficiente para a excitação dos sinais SSVEP;
- Indivíduos testados são pouco sensíveis, biologicamente, à geração das respostas SSVEP;
- Inconsistência do classificador;
- Mau funcionamento da plataforma de aquisição.

Definiram-se então planos de ação para que fossem sanadas as possíveis causas e, em seguida, testada a performance do sistema.

Para a primeira hipótese (desgaste superficial dos eletrodos), verificou-se que, de fato, uma fina camada de óxidos havia se depositado sobre a superfície de alguns eletrodos. A troca foi efetuada, entretanto, a baixa performance persistiu, não sendo possível verificar melhoria no sinal adquirido.

Para a segunda possibilidade (intensidade luminosa insuficiente), a intensidade

dos LEDs foi elevada via software. Assim, como para a primeira hipótese, o sinal adquirido não apresentou melhorias significativas. Entretanto, passou a causar desconforto visual em alguns dos usuários, que relataram cansaço e leve ardência nos olhos.

A terceira possibilidade, embora se mostrasse pouco provável, dado que foram realizados testes com quatro indivíduos diferentes, foi testada. Optou-se pela utilização de outras atividades mentais ou musculares para produzir os sinais que seriam interpretados pelo classificador. As frequências adquiridas foram substituídas por ações como fechar os olhos, mover sutilmente a mão, mover os pés e fechar a mandíbula. As atividades acima foram escolhidas, pois são ações conhecidas por produzirem atributos mais facilmente distinguíveis, sendo esperados, portanto, melhores resultados. O resultado do treinamento apresentou melhorias, como esperado, entretanto, também decaiu consideravelmente na etapa de tempo real. Os resultados para um dos usuários são apresentados na Tabela 4.

Classificação de sinais EEG/EMG alternativos		
	Treinamento	**Testes**
Atividade 1	67%	37%
Atividade 2	89%	62%
Atividade 3	100%	98%
Atividade 4	89%	30%
Atividade 5	50%	26%
Média	83%	61%

Tabela 4 - Resultados de classificação para sinais EEG e EMG alternativo

As hipóteses finais levantadas para a imprecisão do sistema foram o mau funcionamento da plataforma de aquisição ou erros na implementação do classificador. Como forma de testar ambas as hipóteses, foram realizados testes com um dataset de sinais EEG (AVI SSVEP Dataset, disponível em http://www.setzner.com/avi-ssvep-dataset/), disponibilizado por Adnan Vilic, como uma

base de dados livre para uso não-comercial.

O equipamento para a aquisição dos dados foi o g.USBAmp, da fabricante g.Tec com o posicionamento dos eletrodos em Oz, Fz e Fpz segundo o sistema internacional 10-20. O estímulo visual foi gerado com um monitor LCD BenQ XL2420T com taxa de refresh rate de 120Hz, com as frequências de excitação variando entre 6 e 10 Hz. Foram testadas aquisições para cinco sujeitos-teste, tanto do sexo masculino quanto feminino, com idades variando entre 26 e 32 anos. Cada usuário foi exposto a uma sessão composta de 10 trials de 16s cada uma.

Os resultados obtidos foram superiores aos obtidos com os experimentos conduzidos. Três dos usuários cujos dados de aquisição estavam presentes no Dataset utilizado obtiveram elevada taxa de acertos, com o melhor deles atingindo em média 92% de classificações corretas. Além disso, para o pré-processamento dos sinais e extração de atributos, foram utilizadas *epochs* de 1s, valor um quarto menor do que o até então utilizado. Os resultados são apresentados na Tabela 5.

	Usuário 1	Usuário 2	Usuário 3	Usuário 4	Usuário 5
Frequência 1	67%	87%	65%	100%	94%
Frequência 2	50%	100%	0%	50%	75%
Frequência 3	0%	100%	40%	75%	50%
Frequência 4	60%	80%	75%	71%	80%
Frequência 5	62%	100%	69%	82%	100%
Frequência 6	33%	100%	0%	67%	67%
Média	61%	92%	61%	86%	87%

Tabela 5 - Resultados de classificação dos sinais para um dataset de sinais EEG no qual os usuários eram submetidos à indução de sinais SSVEP.

A performance obtida pelo classificador nas condições descritas anteriormente, leva a concluir que um dos maiores fatores que colaborou para o funcionamento impróprio da BCI foi a plataforma de aquisição.

O conforto do usuário foi avaliado de maneira simplificada e adaptada com base nas perguntas presentes nos questionários NASA-TLX (Anexo C), que avaliam o nível de carga percebido pelo usuário durante a realização de alguma tarefa específica. Como os usuários não foram expostos à situações em que precisaram controlar um dispositivo e medir a própria performance, questões relacionadas ao nível de frustração ou dificuldade da tarefa foram omitidas.

As perguntas apresentadas mediam na escala de 0-100 o grau de desgaste físico do usuário, o desgaste psicológico e avaliação do tempo de execução da tarefa. Sendo 0 para baixo nível de desgaste e 100 para alto nível de desgaste.

Informações sobre os usuários					
	Usuário 1	Usuário 2	Usuário 3	Usuário 4	Média
Desgaste Físico	30	40	20	15	35
Desgaste Psicológico	15	15	10	25	15
Tempo de Execução	40	50	30	40	45

Tabela 6 - Avaliação dos usuários com relação ao nível de exigência física, mental e temporal para a utilização da interface. A escala varia de 0 a 100, sendo 0 para baixo desgaste e 100 para desgaste elevado.

7. CONCLUSÕES

O sistema projetado tem potencial para atingir os requisitos propostos, embora não tenha sido possível testar a comunicação entre o dispositivo externo e a interface cérebro-máquina. Considerando o cenário em que o sinal adquirido apresentasse a mesma qualidade dos dados presentes no *dataset* utilizado, o tempo de resposta estaria dentro do tempo limite definido, senão muito próximo, dado que a etapa mais lenta para a geração de uma resposta da BCI é o processamento e classificação dos sinais. Como a utilização de *epochs* de 1s resultou em classificações com bom grau de acurácia para a maioria dos indivíduos analisados, também pode-se inferir que o requisito referente ao tamanho dos pacotes de dados analisados mostrou-se cumprido. No aspecto geral, pode-se presumir que o aprendizado e familiarização do usuário com o sistema possibilitariam o controle de um dispositivo externo.

Como a utilização dos sinais SSVEP prevê a adição de um novo estado mental para cada diferente frequência utilizada, os graus de liberdade sob controle do usuário poderiam ser ampliados sem grandes dificuldades, embora o número de estímulos visuais também esteja sujeita a limitações de processamento e às faixas ideais de operação no espectro de frequências.

Em se tratando de conforto, embora a maior preocupação fosse relativa ao desgaste psicológico proporcionado pelas tarefas a serem executadas durante o treinamento e testes, o nível de desconforto em relação ao tempo de treinamento mostrou-se bastante relevante, o que amplificava também o leve desconforto físico dos usuários diante do equipamento utilizado. Poderia estudar-se a redução do tempo de cada trial ou a redução do número de trials executadas para cada frequência.

Todos os usuários relataram certo desconforto em relação ao headset utilizado, principalmente em função das pontas dos eletrodos secos que produziam marcas e dores temporárias nos locais de contato. Em alguns momentos foi relatado também desconforto visual e leve ardência nos olhos com relação aos LEDs que serviam de estímulo visual. Ainda assim, como os valores médios das notas atribuídas pelos

usuários não foi elevada para nenhum dos critérios (todas as notas abaixo de 50), pode-se dizer que o desconforto não foi excessivo ou prejudicaria a performance durante a execução de alguma tarefa.

Em trabalhos futuros, para a etapa de pré-processamento dos sinais pode-se estudar a utilização de algoritmos de tratamento de sinais mais sofisticados. Enquanto isso, a classificação dos atributos pode ser avaliada sob um viés voltado para *feature engineering,* de forma a utilizar de outros atributos derivados das amplitudes resultantes do cálculo da FFT do sinal. Em algumas pesquisas, verificou-se, por exemplo, a utilização de bandas mais largas do espectro para computar a energia média de cada uma das bandas. A geração dos estímulos visuais também poderia ser feito via monitor LCD, desde que o aparelho conte com uma taxa de *refresh* suficientemente alta e o computador apresente boa performance de processamento gráfico. A utilização de interfaces como PsychToolbox, possibilita um setup muito mais rápido e configurações mais versáteis do que a abordagem por LEDs.

Por fim, embora não tenham sido realizados testes de integração com dispositivos externos, espera-se que o aprendizado gerado com este trabalho possa embasar futuros projetos do mesmo tema no Laboratório de Biomecatrônica.

8. REFERÊNCIAS

A. B. A. Graf and R. A. Andersen. “Brain–machine Interface for Eye Movements.” Proceedings of the National Academy of Sciences of the United States of America 111.49 (2014): 17630–17635. PMC. Web. 29 Oct. 2016. Disponível em: <http://www.pnas.org/content/111/49/17630.abstract>

A. El-Madani, H. B. Sorensen, T. W. Kjær, C. E. Thomsen, and S. Puthusserypady, "Real-time brain computer interface using imaginary movements," EPJ Nonlinear Biomedical Physics, journal article vol. 3, no. 1, p. 9, 2015.

A. Ferreira, W. C. Celeste, F. A. Cheein, T. F. Bastos-Filho, M. Sarcinelli-Filho, & R. Carelli (2008). Human-machine interfaces based on EMG and EEG applied to robotic systems. Journal of NeuroEngineering and Rehabilitation, 5, 10. Disponível em: <http://doi.org/10.1186/1743-0003-5-10>

A. Güneysu, H. L. Akin, "An SSVEP based BCI to control a humanoid robot by using portable EEG device," *2013 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, Osaka, 2013, pp. 6905-6908. Diponível em: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6611145&isnumber=6609410>

E. Hortal, D. Planelles, A. Costa, E. Iáñez, A. Úbeda, J.M. Azorín, E. Fernández, SVM-based Brain–Machine Interface for controlling a robot arm through four mental tasks, Neurocomputing, Volume 151, Part 1, 3 March 2015, Pages 116-121, ISSN 0925-2312. Disponível em: <http://www.sciencedirect.com/science/article/pii/S092523121401323X>

E. Hortal, A. Úbeda, E. Iáñez, J. M. Azorín, Control of a 2 DoF robot using a Brain–Machine Interface, Computer Methods and Programs in Biomedicine, Volume 116, Issue 2, September 2014, Pages 169-176, ISSN 0169-2607. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0169260714000856>

E. Hortal, E. Iáñez, A. Úbeda, C. Perez-Vidal, J. M. Azorín, Combining a Brain–Machine Interface and an Electrooculography Interface to perform pick and place tasks with a robotic arm, Robotics and Autonomous Systems, Volume 72, October 2015, Pages 181-188, ISSN 0921-8890. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0921889015001244>

F. Afdideh, M. B. Shamsollahi and S. N. Resalat, "Development of a MATLAB-based toolbox for brain computer interface applications in virtual reality," 20th Iranian Conference on Electrical Engineering (ICEE2012), Tehran, 2012, pp. 1579-1583. Disponível em: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6292612&isnumber=6292310>

F. Galán, M. Nuttin, E. Lew, P.W. Ferrez, G. Vanacker, J. Philips, J. del R. Millán, A brain-actuated wheelchair: Asynchronous and non-invasive Brain–computer interfaces for continuous control of robots, Clinical Neurophysiology, Volume 119, Issue 9, September 2008, Pages 2159-2169, ISSN 1388-2457, Disponível em: <http://www.sciencedirect.com/science/article/pii/S1388245708005750>

J. R. Millan, F. Renkens, J. Mourino and W. Gerstner, "Noninvasive brain-actuated control of a mobile robot by human EEG," in *IEEE Transactions on Biomedical Engineering*, vol. 51, no. 6, pp. 1026-1033, June 2004. Disponível em: < http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1300798&isnumber=28897>

J. Dabrowski, E. V. Munson; 40 years of searching for the best computer system response time. Interact Comput 2011; 23 (5): 555-564. doi: 10.1016/j.intcom.2011.05.008. <Disponível em: https://academic.oup.com/iwc/article/23/5/555/662569/40-years-of-searching-for-the-best-computer-system>

J. R. Wolpaw, N. Birbaumer, D. J. McFarland, G. Pfurtscheller, T. M Vaughan, “Brain–computer interfaces for communication and control, In Clinical Neurophysiology”, Volume 113, Issue 6, 2002, Pages 767-791, ISSN 1388-2457, Disponível em: <http://www.sciencedirect.com/science/article/pii/S1388245702000573>

K. T. Kim, T. Carlson and S. W. Lee, "Design of a robotic wheelchair with a motor imagery based brain-computer interface," Brain-Computer Interface (BCI), 2013 International Winter Workshop on, Gangwo, 2013, pp. 46-48. Disponível em: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6506625&isnumber=6506604>

K. Takahashi and T. Kashiyama, "Remarks on multimodal nonverbal interface and its application to controlling mobile robot," IEEE International Conference Mechatronics and Automation, 2005, 2005, pp. 999-1004 Vol. 2. Disponível em: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1626688&isnumber=34147>

K. Tanaka, K. Matsunaga and H. O. Wang, "Electroencephalogram-Based Control of an Electric Wheelchair," in IEEE Transactions on Robotics, vol. 21, no. 4, pp. 762-766, Aug. 2005. Disponível em: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1492493&isnumber=32079>

L. Bueno, J. L. Pons and T. F. Bastos Filho, "An embedded system for an EEG based BCI," Biosignals and Biorobotics Conference (BRC), 2013 ISSNIP, Rio de Janeiro, 2013, pp. 1-5. Disponível em: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6487457&isnumber=6487444>

Maria A. Pastor, Julio Artieda, Javier Arbizu, Miguel Valencia and Jose C. Masdeu, “Human Cerebral Activation druing Steady-State Visual-Evoked Responses”, Journal of Neuroscience, December 2003. Disponível em: <http://www.jneurosci.org/content/23/37/11621>

M. Middendorf, G. McMillan, G. Calhoun and K. S. Jones, "Brain-computer interfaces based on the steady-state visual-evoked response," in *IEEE Transactions on Rehabilitation Engineering*, vol. 8, no. 2, pp. 211-214, Jun 2000. <Disponível em: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=847819&isnumber=18398>

R. B. Miller, 1968 , Response time in man-computer conversational transactions . Proceedings Spring Joint Computer Conference 1968 , 33 (Montvale , New Jersey AFIPS Press), pp. 267 – 277 .<Disponível em: http://theixdlibrary.com/pdf/Miller1968.pdf>

S. Parini, L. Maggi, , A. C. Turconi and G. Andreoni, “A Robust and Self-Paced BCI System Based on a Four Class SSVEP Paradigm: Algorithms and Protocols for a High-Transfer-Rate Direct Brain Communication”. *Computational Intelligence and Neuroscience*, *2009*, 864564. Disponível em: <http://doi.org/10.1155/2009/864564>

APÊNDICE A: CÓDIGO-FONTE

Arquivo: processa_dados_eeg_scores.py

Finalidade: Recebe os dados brutos de EEG, os filtra e analisa as frequências de pico da fft, retornando scores que comparam a frequência real e a do resultado.

```
1. # -*- coding: utf-8 -*-
2. """
3. processa_daados_eeg_scores.py
4.
5. @author: Rafael and Lucas
6. """
7. import pandas as pd
8. import matplotlib.pyplot as plt
9. import plotly
10. import scipy.signal
11. #import kalman
12. import pywt
13. import numpy
14. import scipy
15. #import pywt
16.                         plotly.tools.set_credentials_file(username='RafaelPalmerio',
    api_key='pMHFCPMBYUmaK9PYhx3h')
17. import numpy as np
18. # Learn about API authentication here: https://plot.ly/python/getting-started
19. # Find your api_key here: https://plot.ly/settings/api
20.
21.
22.
23. def get_results(arquivo, amostra, freqs, debug=False):
24.     numero =0
25.     lista_index=[]
26.     df = pd.read_csv(arquivo)
27.     df = df.iloc[1000:-1000].reset_index()
28.     divisao_minima = 0.25
29.     tamanho_janela_segundos = 4
30.
31.     tam_janela = int(tamanho_janela_segundos/divisao_minima)
32.
33.     for index,row in df.iterrows():
34.         if numero>=250*divisao_minima:
35.             numero=1
36.         else:
37.             numero+=1
38.         lista_index.append(numero)
39.     df['index_num'] = lista_index
40.     #print(inicio, fim+1)
41.
42.     # faz a media de valor lido por index
43.     #soma = df.groupby('index').mean()
44.     # define quais valores serão usados na fft
45.     #y = df[[canal,'index']].drop_duplicates('index', keep='last')[canal]
46.     #y = df[canal].iloc[2000:2256]
47.     #y = soma[canal]
48.
49.     def media_movel(df, n=100):
50.         """
51.         Faz a média móvel do sinal
```

```
        """
        df['dummy'] = df.index//n
        df['media'] = df.groupby('dummy').mean()
                                                                    df        =
df.drop('media',axis=1).merge(df[['media']].dropna().reset_index(),left_on='dummy',
right_on='index').\
                                    drop(['dummy', 'index'],axis=1).rename(columns =
{'final':'final_0'})
        df['final'] = df['final_0'] - df['media']
        return df[['final']]


    def pega_janelas(inicio, fim,df,canal):
         # divide o sinal em janelas do tempo especificado no inicio pelo seu valor
de index
        numero=0
        lista_numero = []
        for index,row in df.iterrows():
            if row['index_num'] ==1:
                numero+=1
            lista_numero.append(numero)
        df['numero'] = lista_numero
        #print(inicio, fim+1)
        df = df[df['numero'].isin(range(inicio,fim+1))]
        soma = df.groupby('index_num').mean()
        y = soma[canal]
        return df,y

    def soma_canais(df,grupo_1, grupo_2, media=True):
        grupo_1 = list(map(lambda x: 'canal_' + str(x),grupo_1))
        grupo_2 = list(map(lambda x: 'canal_' + str(x),grupo_2))
        df = df[grupo_1+grupo_2]
        if media:
            df = df - df.mean()
        df_grupo_1 = df[grupo_1].mean(axis=1)
        df_grupo_2 = df[grupo_2].mean(axis=1)
        return df_grupo_1-df_grupo_2


    Wn = [4/125,20/125]
    Wn =0.16
    #aplica o filtro de butterworth
    def butterworth(y, Wn=20/125,N=5, tipo = 'high'):
        b, a = scipy.signal.butter(N, Wn, tipo)
        y = scipy.signal.filtfilt(b, a, y)
        return y

    #Fs = 255.0;  # sampling rate

    def fft(Fs,y):
        Ts = 1.0/Fs; # sampling interval

           #ff = 5;   # frequency of the signal
           #y = np.sin(2*np.pi*ff*t)
           #aplica a fft no sinal filtrado
           t = np.arange(0,Ts*len(y),Ts) # time vector


           n = len(y) # length of the signal
           if n % 2 ==1:
               n=n-1
```

```
109.            #print(n)
110.            k = np.arange(n)
111.            #print(k)
112.            T = n/Fs
113.            frq = k/T # two sides frequency range
114.            frq = frq[range(n//2)] # one side frequency range
115.
116.            Y = np.fft.fft(y)/n # fft computing and normalization
117.            Y = Y[range(n//2)]
118.            return frq,abs(Y),t
119.
120.
121.        def hanning(y):
122.            han = np.hanning(y.shape[0])
123.            return han
124.
125.        def blackman(y):
126.            return np.blackman(y.shape[0])
127.
128.        def hamming(y):
129.            ham = np.hamming(y.shape[0])
130.            return ham
131.
132.        def flattop(y, sym=True):
133.            ft = scipy.signal.flattop(y.shape[0], sym=sym)
134.            return ft
135.
136.        def subtrai_media(y):
137.            return y-y.mean()
138.
139.        def wl(y,name):
140.            cA, cD = pywt.dwt(y, name)
141.            #y2 = pywt.idwt(cA, cD, name)
142.            return cA,cD
143.
144.        def wavelet(y,name):
145.            noiseSigma = 0.16
146.            levels  = int( np.floor( np.log2(y.shape[0]) ) )-3
147.            coeffs = pywt.wavedec(data=y, wavelet=name, level=levels)
148.            threshold = noiseSigma*np.sqrt(2*np.log2(y.size))
149.                 NewWaveletCoeffs = list(map (lambda x: pywt.threshold(x,threshold,
    mode='soft'),
150.                                      coeffs))
151.            #print(NewWaveletCoeffs)
152.            #print(levels)
153.            New_y = pywt.waverec( NewWaveletCoeffs, wavelet=name)
154.            return New_y
155.
156.        def get_output_array(amostra, tipo = 'numerico'):
157.            # fazo vetor saída baseado na key do dicionario
158.            if tipo == 'vetorial':
159.                output = np.array([[0,0,0,0,0]])
160.                if '7' in amostra:
161.                    output[0][0] = 1
162.                elif '9' in amostra:
163.                    output[0][1] = 1
164.                elif '11' in amostra:
165.                    output[0][2] = 1
166.                elif '13' in amostra:
167.                    output[0][3] = 1
168.                elif '15' in amostra:
```

```
                    output[0][4] = 1
            elif tipo == 'numerico':
                if '7' in amostra:
                    output = np.array([0])
                elif '9' in amostra:
                    output = np.array([1])
                elif '11' in amostra:
                    output = np.array([2])
                elif '13' in amostra:
                    output = np.array([3])
                elif '15' in amostra:
                    output = np.array([4])
            elif tipo == 'resposta':
                output = int(amostra.split('_')[0])
            return output

        def filtra_apply(row):
            for freq in freqs:
                                                if   row['frequencia_olhos']<freq+1   and
    row['frequencia_olhos']>freq-1:
                    return freq
            return None


        #define o canal de leitura
        canal='canal_1'


        janelas = df[df.index_num==1].shape[0]

        # df é o df original, z é só o canal especificado
        df, z=pega_janelas(1,janelas,df,canal)


        #divide os canais em grupos pela proximidade física
        grupo_1 = [1,2,3,4,5]
        grupo_2 = [6]

        #soma os canais pelos grupos especificados acima
        df['final'] = soma_canais(df, grupo_1, grupo_2, media= True)

        #inicializando os containers de dados
        results_matrix = np.matlib.zeros((freqs.__len__(), freqs.__len__()))

        #pegando um canal só
        #df['final'] = df[canal]
        for i in range(1,janelas+1):
            #df, y=pega_janelas(i,5,df,canal)
            #y=soma[canal]
            #y=df[canal]

            #y = df[canal][df['numero']==i].reset_index(drop=True)
            y = df[['final']][df['numero'].isin(range(i, i+tam_janela))]\
                                .reset_index(drop=True)
            #y = df[['final']][df]

            if y.__len__() < 250*tamanho_janela_segundos:
                continue


            #y=media_movel(y)
```

```
229.            #y = subtrai_media(y)
230.
231.            y = y['final']
232.            #y=butterworth(y,Wn=0.8, tipo='low')
233.            #y=butterworth(y,Wn=0.07, tipo='high', N=8)
234.
235.            y = y*flattop(y)
236.
237.            y = wavelet(y,'coif1')
238.
239.
240.            #y=butterworth(y,Wn=0.4)
241.
242.            #y = kalman.kalman(y,n_iter=len(y))
243.
244.
245.            #y = y*flat_top(y, False)
246.            #t = np.arange(0,100, 0.01)
247.            #y = np.sin(t)
248.
249.            frq,Y,t=fft(250,y)
250.            #plota a fft do sinal filtrado
251.            #fig, ax = plt.subplots(2, 1)
252.            #ax[0].plot(t,y)
253.            #ax[0].set_xlabel('Time')
254.            #ax[0].set_ylabel('Amplitude')
255.
256.            #t = np.arange(0,100, 0.01)
257.
258.    #         plt.plot(t,y)
259.    #         plt.xlabel('Time')
260.    #         plt.ylabel('Amplitude')
261.    #         plt.show()
262.            #ax[1].plot(frq,abs(Y),'r') # plotting the spectrum
263.            #ax[1].set_xlabel('Freq (Hz)')
264.            #ax[1].set_ylabel('|Y(freq)|')
265.
266.            #plot_url = py.plot_mpl(fig, filename='mpl-basic-fft')
267.
268.            # pega as frequencias de maior amplitude na fft do sinal filtrado
269.            kappa = pd.DataFrame()
270.            kappa['frequencia_olhos'] = frq
271.            kappa['amplitude'] = Y
272.
273.            kappa = kappa.iloc[0*tamanho_janela_segundos:20*tamanho_janela_segundos]
274.            output_desejado = int(get_output_array(amostra, tipo='resposta'))
275.
276.            kappa['frequencia'] = kappa.apply(filtra_apply, axis=1)
277.
278.            kappa = kappa[~kappa['frequencia'].isnull()].reset_index(drop=True)
279.
280.            maximo = kappa[kappa.amplitude==kappa.amplitude.max()].iloc[0]
281.            output_real = int(maximo['frequencia'])
282.
283.            row_index = freqs.index(output_desejado)
284.            col_index = freqs.index(output_real)
285.            #print(maximo)
286.            #print('Máximo: ',fr_max, 'Resposta: ', output )
287.            results_matrix[row_index, col_index] += 1
288.            if debug:
289.                                        print(kappa[(kappa['frequencia_olhos']<17)  &
```

```
    (kappa['frequencia_olhos']>6)]\
290.
    .sort_values('amplitude',ascending=False).reset_index(drop=True).iloc[0:40])
291.       #return dataset, final_output
292.       return results_matrix
293.
294.
295.                                                                  filenames        =
    {'8_0':r'C:\Users\Rafael\Documents\TCC\BCI\EEG_Data\OpenBCI-RAW-teste_8hz.csv',
296.
    '12_0':r'C:\Users\Rafael\Documents\TCC\BCI\EEG_Data\OpenBCI-RAW-teste_12hz.csv',
297.
    '10':r'C:\Users\Rafael\Documents\TCC\BCI\EEG_Data\OpenBCI-RAW-teste_olhos_fechados.
    csv',
298.
    '8_1':r'C:\Users\Rafael\Documents\TCC\BCI\EEG_Data\OpenBCI-RAW-20170918_8hz.csv',
299.
    '15_0':r'C:\Users\Rafael\Documents\TCC\BCI\EEG_Data\OpenBCI-RAW-20170918_15hz.csv',
300.
    '7':r'C:\Users\Rafael\Documents\TCC\BCI\EEG_Data\OpenBCI-RAW-20170918_7hz.csv',
301.                                                                    '7_1':
    r'C:\Users\Rafael\Documents\TCC\BCI\EEG_Data\OpenBCI-RAW-20170921_7hz.csv',
302.
    '9':r'C:\Users\Rafael\Documents\TCC\BCI\EEG_Data\OpenBCI-RAW-20170921_9hz.csv',
303.
    '11':r'C:\Users\Rafael\Documents\TCC\BCI\EEG_Data\OpenBCI-RAW-20170921_11hz.csv',
304.
    '13':r'C:\Users\Rafael\Documents\TCC\BCI\EEG_Data\OpenBCI-RAW-20170921_13hz.csv',
305.
    '15_1':r'C:\Users\Rafael\Documents\TCC\BCI\EEG_Data\OpenBCI-RAW-20170921_15hz.csv',
306.
    '7_2':r'C:\Users\Rafael\Documents\TCC\BCI\EEG_Data\OpenBCI-RAW-20171005_7hz.csv',
307.
    '7_3':r'C:\Users\Rafael\Documents\TCC\BCI\EEG_Data\OpenBCI-RAW-20171005_7hz_2.csv',
308.
    '9_2':r'C:\Users\Rafael\Documents\TCC\BCI\EEG_Data\OpenBCI-RAW-20171005_9hz.csv',
309.
    '9_3':r'C:\Users\Rafael\Documents\TCC\BCI\EEG_Data\OpenBCI-RAW-20171005_9hz_2.csv',
310.
    '11_2':r'C:\Users\Rafael\Documents\TCC\BCI\EEG_Data\OpenBCI-RAW-20171005_11hz.csv',
311.
    '11_3':r'C:\Users\Rafael\Documents\TCC\BCI\EEG_Data\OpenBCI-RAW-20171005_11hz_2.csv
    ',
312.
    '13_2':r'C:\Users\Rafael\Documents\TCC\BCI\EEG_Data\OpenBCI-RAW-20171005_13hz.csv',
313.
    '13_3':r'C:\Users\Rafael\Documents\TCC\BCI\EEG_Data\OpenBCI-RAW-20171005_13hz_2.csv
    ',
314.
    '15_2':r'C:\Users\Rafael\Documents\TCC\BCI\EEG_Data\OpenBCI-RAW-20171005_15hz.csv',
315.
    '15_3':r'C:\Users\Rafael\Documents\TCC\BCI\EEG_Data\OpenBCI-RAW-20171005_15hz_2.csv
    '
316.                       }
317.
318.
319.    usaveis = ['15_0', '7', '7_1', '9', '11', '13', '15_1']
320.     usaveis = [ '7_2', '7_3', '9_2', '9_3', '11_2', '11_3','13_2', '13_3', '15_2',
    '15_3']
321.
322.    usaveis = [ '7','7_1','7_2', '7_3', '9', '9_2', '9_3','11', '11_2', '11_3',
```

```
            '13','13_2','13_3','15_0', '15_2', '15_3']

#usaveis = [ '7_2', '7_3','11_2', '11_3',\
#           '13_2', '13_3', '15_2', '15_3']

#usaveis = [ '7_2', '7_3', '9_2', '9_3', \
#            '13_2', '13_3', '15_2', '15_3']

freqs = list(set(list(map(lambda x: int(x.split('_')[0]), usaveis))))
results_matrix = np.matlib.zeros((freqs.__len__(), freqs.__len__()))
for index, amostra in enumerate(usaveis):
    print(amostra)
      results_matrix = results_matrix + get_results(filenames[amostra], amostra, freqs)
    """
    if index == 0:
        final_dataset = kap
        final_output = output
    else:
        final_dataset = (np.concatenate((final_dataset,
                                         kap), axis=0))
        final_output = (np.concatenate((final_output,
                                         output), axis=0))
    """

def recall_precision(matrix):
    size = matrix.shape[0]
    acertos = 0
    df = pd.DataFrame(matrix)
    score_df = pd.DataFrame(columns=['precision', 'recall', 'support'])
    for line in range(size):
        acertos += df[line].iloc[line]
        row = {}
        row['precision'] = df[line].iloc[line]/df[line].sum()
        row['recall'] = df[line].iloc[line]/df.iloc[line].sum()
        row['support'] = int(df.iloc[line].sum())
        score_df = score_df.append(row, ignore_index=True)
    score_df['support'] = score_df['support'].astype(int)
    avg_total = score_df.mean()
    avg_total['support'] = int(score_df['support'].sum())
    avg_total['overall_precision'] = acertos/avg_total['support']
    return score_df, avg_total

confusion_matrix = recall_precision(results_matrix)

print(results_matrix)
print()
print(confusion_matrix[0])
print()
print(confusion_matrix[1])
```

Arquivo: processa_dados_eeg_neural_network_tempo_real.py

Finalidade: Recebe os dados brutos de EEG, os filtra e coloca em matrizes no formato certo para os classificadores.

```
1. # -*- coding: utf-8 -*-
2. """
3. processa_dados_eeg_neural_network_tempo_real.py
4.
5. @author: Rafael and Lucas
6. """
7. import pandas as pd
8. import matplotlib.pyplot as plt
9. import plotly
10. import scipy.signal
11. import kalman
12. import neural_network
13. import pywt
14. import numpy
15. import scipy
16. #import pywt
17.                       #plotly.tools.set_credentials_file(username='RafaelPalmerio',
    api_key='pMHFCPMBYUmaK9PYhx3h')
18. import numpy as np
19.
20. ###############################
21. #       Processamento         #
22. ###############################
23.
24. def media_movel(df, n=128, n_canais=8):
25.     """
26.     Faz a média móvel do sinal
27.     """
28.     # colunas dos valores
29.     canais = ['canal_' + str(i) for i in range(1,n_canais+1)]
30.     colunas_medias = ['media_' + str(i) for i in range(1,n_canais+1)]
31.
32.     # fazendo a coluna dummy para agrupamento das linhas
33.     temp = df.copy()
34.     temp['dummy'] = temp.index//n
35.
36.     # agrupando pelo dummy  e renomeando as colunas
37.         media  =  temp.groupby('dummy').mean().rename(columns  =  {a:b  for  a,b  in
    zip(canais, colunas_medias)}).reset_index()
38.     media = media[['dummy'] + colunas_medias]
39.
40.     # left join para pegar os valores de media
41.     temp = temp.merge(media, on='dummy', how='left')
42.
43.     # tirando o valor da media de cada canal em cada janela
44.     for canal, media in zip(canais, colunas_medias):
45.         temp[canal] -= temp[media]
46.
47.     # dropando as colunas auxiliares
48.     temp = temp.drop(colunas_medias + ['dummy'], axis=1)
49.
50.     return temp
51.
52. def pega_janelas(inicio, fim,df,canal):
```

```
    """
     divide o sinal em janelas do tempo especificado no inicio pelo seu valor de index
    """
    numero=0
    lista_numero = []
    for index,row in df.iterrows():
        if row['index_num'] ==1:
            numero+=1
        lista_numero.append(numero)
    df['numero'] = lista_numero
    #print(inicio, fim+1)
    df = df[df['numero'].isin(range(inicio,fim+1))]
    soma = df.groupby('index_num').mean()
    y = soma[canal]
    return df,y

def soma_canais(df,grupo_1, grupo_2, media=False):
    """
    soma os grupos de canais pela sua localização na cabeça
    """
    grupo_1 = ['canal_' + str(x) for x in grupo_1]
    grupo_2 = ['canal_' + str(x) for x in grupo_2]
    df = df[grupo_1+grupo_2]

    if media:
        df = df - df.mean()
    df_grupo_1 = df[grupo_1].sum(axis=1)
    df_grupo_2 = df[grupo_2].sum(axis=1)
    res = (df_grupo_1**2-df_grupo_2)

    return res

def tira_pontas(df, s_comeco, s_fim):
    """
    Funcção que tira s_comeco segundos do comeco do dataframe e s_fim do final
    """
    lista = list(df.index[df['index_num']==1])
    index_inicio = lista[s_comeco]
    index_fim = lista[-s_fim]
    df = df.iloc[index_inicio:index_fim]
    return df

def normalize_channels(df, n_canais=8):
    cols = ['canal_'+str(i) for i in range(1,n_canais+1)]
    for col in cols:
        if df[col].std() != 0:
            df[col]=(df[col]-df[col].mean())/df[col].std()
       return df

   def fft(Fs,y):
       """
       faz a fft do sinal
       """
       # tamanho do sinal
       n = len(y)
       if n % 2 == 1:
           n=n-1
           # ajustando o tamanho do vetor para coincidirem os tamanhos
           y = y[:-1]

```

```
113.        Ts = 1.0/Fs; # sampling interval
114.
115.        #aplica a fft no sinal filtrado
116.        t = np.arange(0,Ts*len(y),Ts) # time vector
117.
118.        #print(n)
119.        k = np.arange(n)
120.        #print(k)
121.        T = n/Fs
122.        frq = k/T # two sides frequency range
123.        frq = frq[range(n//2)] # one side frequency range
124.
125.        Y = np.fft.fft(y)/n # fft computing and normalization
126.        Y = Y[range(n//2)]
127.        return frq,abs(Y),t
128.
129.    def butterworth(y, Wn=20/125,N=5, tipo = 'high'):
130.        """
131.        aplica o filtro de butterworth
132.        """
133.        b, a = scipy.signal.butter(N, Wn, tipo)
134.        y = scipy.signal.filtfilt(b, a, y)
135.        return y
136.
137.    def filtra_sinal(y, high_Wn = 0.1, high_N = 8,
138.                                        low_Wn=0.1, low_N = 8, hann=False, hamm=False,
    fltop=False, wvlt=False):
139.        """
140.        função que aplica os filtros de maneira uniforme
141.        """
142.        if low_Wn:
143.            y=butterworth(y,Wn=low_Wn, tipo='low', N=low_N)
144.        if high_Wn:
145.            y=butterworth(y,Wn=high_Wn, tipo='high', N=high_N)
146.
147.        if hann:
148.            y = y*hanning(y)
149.
150.        if hamm:
151.            y = y*hamming(y)
152.
153.        if fltop:
154.            y = y*flattop(y)
155.
156.        if wvlt:
157.            y = wavelet(y,'coif1')
158.        #y = kalman.kalman(y,n_iter=len(y))
159.
160.        return y
161.
162.    def filtra_sinal_todo(df, high_Wn = 0.1, high_N = 8,
163.                                        low_Wn=0.1, low_N = 8, hann=False, hamm=False,
    fltop=False, wvlt=False, n_canais=8):
164.        """
165.        função que aplica os filtros de maneira uniforme
166.        """
167.        cols = ['canal_'+str(i) for i in range(1,9)]
168.        for col in cols:
169.
170.            if low_Wn:
171.                df[col]=butterworth(df[col],Wn=low_Wn, tipo='low', N=low_N)
```

```
172.             if high_Wn:
173.                 df[col]=butterworth(df[col],Wn=high_Wn, tipo='high', N=high_N)
174.
175.             if hann:
176.                 df[col] = df[col]*hanning(df[col])
177.
178.             if hamm:
179.                 df[col] = df[col]*hamming(df[col])
180.
181.             if fltop:
182.                 df[col] = df[col]*flattop(df[col])
183.
184.             if wvlt:
185.                 df[col] = wavelet(df[col],'coif1')
186.         #y = kalman.kalman(y,n_iter=len(y))
187.
188.         return df
189.
190.     def get_output_array(amostra, tipo = 'numerico', source = 'tempo_real'):
191.         """
192.         faz o vetor saída baseado na key do dicionario
193.         """
194.         if source != 'download':
195.             if tipo == 'vetorial':
196.                 output = np.array([[0,0,0,0,0]])
197.                 if '100':
198.                     output[0][5] = 1
199.                 elif '7' in amostra:
200.                     output[0][0] = 1
201.                 elif '9' in amostra:
202.                     output[0][1] = 1
203.                 elif '11' in amostra:
204.                     output[0][2] = 1
205.                 elif '13' in amostra:
206.                     output[0][3] = 1
207.                 elif '15' in amostra:
208.                     output[0][4] = 1
209.             elif tipo == 'numerico':
210.                 if '100' in amostra:
211.                     output = np.array([5])
212.                 elif '7' in amostra:
213.                     output = np.array([0])
214.                 elif '9' in amostra:
215.                     output = np.array([1])
216.                 elif '11' in amostra:
217.                     output = np.array([2])
218.                 elif '13' in amostra:
219.                     output = np.array([3])
220.                 elif '15' in amostra:
221.                     output = np.array([4])
222.         else:
223.             output = np.array([int(amostra)])
224.         return output
225.
226.     def plot_fft(frq,Y,t,y):
227.         #plota a fft do sinal filtrado
228.         fig, ax = plt.subplots(2, 1)
229.         ax[0].plot(t,y)
230.         ax[0].set_xlabel('Time')
231.         ax[0].set_ylabel('Amplitude')
232.
```

```
233.        plt.plot(t,y)
234.
235.        plt.xlabel('Time')
236.        plt.ylabel('Amplitude')
237.        plt.show()
238.        plt.plot(frq,Y)
239.        ax[1].plot(frq,abs(Y),'r') # plotting the spectrum
240.        ax[1].set_xlabel('Freq (Hz)')
241.        ax[1].set_ylabel('|Y(freq)|')
242.        plt.show()
243.
244.        #plot_url = py.plot_mpl(fig, filename='mpl-basic-fft')
245.
246.    ############################
247.    #       Windowing          #
248.    ############################
249.
250.    def hanning(y):
251.        han = np.hanning(y.shape[0])
252.        return han
253.
254.    def blackman(y):
255.        return np.blackman(y.shape[0])
256.
257.    def hamming(y):
258.        ham = np.hamming(y.shape[0])
259.        return ham
260.
261.    def flattop(y, sym=True):
262.        ft = scipy.signal.flattop(y.shape[0], sym=sym)
263.        return ft
264.
265.    def subtrai_media(y):
266.        return y-y.mean()
267.
268.    def wl(y,name):
269.        cA, cD = pywt.dwt(y, name)
270.        #y2 = pywt.idwt(cA, cD, name)
271.        return cA,cD
272.
273.    def wavelet(y,name):
274.        noiseSigma = 0.16
275.        levels  = int( np.floor( np.log2(y.shape[0]) ) )-3
276.        coeffs = pywt.wavedec(data=y, wavelet=name, level=levels)
277.        threshold = noiseSigma*np.sqrt(2*np.log2(y.size))
278.                NewWaveletCoeffs  =  list(map  (lambda  x:  pywt.threshold(x,threshold,
    mode='soft'),
279.                                    coeffs))
280.        #print(NewWaveletCoeffs)
281.        #print(levels)
282.        New_y = pywt.waverec( NewWaveletCoeffs, wavelet=name)
283.        return New_y
284.
285.    ##########################################################################
286.    #       Processa uma amostra vinda de aquisição em tempo real            #
287.    ##########################################################################
288.
289.    def processa_real_time(df, tamanho_janela_segundos=4, high_Wn = 0.1, high_N = 8,
290.                                        low_Wn=0.1,  low_N  =  8,  hann=False,  hamm=False,
    fltop=False, wvlt=False):
291.        #media movel
```

```
292.        df = media_movel(df)
293.
294.        # divide o sinal em dois grupos
295.        grupo_1 = [1,2,3,4,5]
296.        grupo_2 = [6]
297.
298.        #soma os canais pelos grupos especificados acima
299.        df['final'] = soma_canais(df, grupo_1, grupo_2, media= True)
300.
301.        #y=soma[canal]
302.        #y=df[canal]
303.
304.        #y = df[['final']][df]
305.
306.        #y = subtrai_media(y)
307.
308.        y = df['final']
309.
310.        y = filtra_sinal(y, high_Wn = high_Wn, high_N = high_N,
311.                                    low_Wn=low_Wn, low_N = low_N, hann=hann, hamm=hamm,
    fltop=fltop, wvlt=wvlt)
312.
313.
314.        #y = y*flat_top(y, False)
315.        frq,Y,t=fft(float(y.shape[0])/tamanho_janela_segundos,y)
316.        #frq,Y,t=fft(float(250,y)
317.        #plot_fft()
318.
319.        # pega as frequencias de maior amplitude na fft do sinal filtrado
320.        kappa = pd.DataFrame()
321.        kappa['frequencia_olhos'] = frq
322.        kappa['amplitude'] = Y
323.
324.        kappa = kappa.iloc[0*tamanho_janela_segundos:25*tamanho_janela_segundos]
325.
326.        dataset = np.array(kappa[['amplitude']]).transpose()
327.
328.                                            #print(kappa[(kappa['frequencia_olhos']<17)    &
    (kappa['frequencia_olhos']>6)]\
329.                                                                                          #
    .sort_values('amplitude',ascending=False).reset_index(drop=True).iloc[0:40])
330.        return dataset
331.
332.
333.
334.    ################################################
335.    #       Processa uma amostra de arquivos       #
336.    ################################################
337.
338.    def processa_from_file(arquivo, amostra, divisao_minima = 1,
339.                               tamanho_janela_segundos = 4, source = 'tempo_real',
340.                               s_comeco = 2, s_fim = 2,
341.                               high_Wn = 0.1, high_N = 8,
342.                                          low_Wn=0.1, low_N = 8, hann=False, hamm=True,
    fltop=False, wvlt=False):
343.        # fft separada: fft é feita por canal e seus vetores são juntados
344.        fft_separada = True
345.
346.        # junto: o processamento do sinal é feito no sinal todo, e não nas janelas
347.        junto=False
348.        if junto and fft_separada:
```

```
            assert False
        aux_bool = True
        freq_aq = 250
        n_canais = 8

        numero =0
        lista_index=[]
        df = pd.read_csv(arquivo)


        # para cada caso, as frequencias podem ser diferentes
        if source == 'download':
            if df.shape[1] > 2:
                freq_aq = 128
                n_canais=14
            else:
                freq_aq = 512
                n_canais=1

        #df = normalize_channels(df, n_canais=n_canais)
        df = media_movel(df, n_canais=n_canais)


        if not source == 'tempo_real':
            for index,row in df.iterrows():
                if numero>=freq_aq*divisao_minima:
                    numero=1
                else:
                    numero+=1
                lista_index.append(numero)
            df['index_num'] = lista_index


        if source != 'download':
            df = tira_pontas(df, s_comeco, s_fim)
        else:
            #df = tira_pontas(df, 1, 1)
            pass
        tam_janela = int(tamanho_janela_segundos/divisao_minima)

        #print(inicio, fim+1)

        # faz a media de valor lido por index
        #soma = df.groupby('index').mean()

        #y = soma[canal]

        #define o canal de leitura
        canal='canal_1'


        janelas = df[df.index_num==1].shape[0]
        # df é o df original, z é só o canal especificado
        df, z=pega_janelas(1,janelas,df,canal)

        #divide os canais em grupos pela proximidade física
        if source == 'download' and n_canais == 14:
            grupo_1 = [5,6,7,8,9,10]
            grupo_2 = [1,2,3,4,11,12,13,14]
        else:
            grupo_1 = [1,2,3,4,5]
```

```
            grupo_2 = [6]
        #soma os canais pelos grupos especificados acima
        if not junto and not fft_separada:
            df['final'] = soma_canais(df, grupo_1, grupo_2, media= False)



#frq,Y,t=fft(float(df['final'].shape[0])/tamanho_janela_segundos,df['final'])
        #plot_fft(frq,Y,t,df['final'])

        #inicializando os containers de dados
        dataset = []
        final_output = []

        #pegando um canal só
        #df['final'] = df[canal]
        for i in range(1,janelas+1):
            #y=soma[canal]
            #y=df[canal]
            #y = df[canal][df['numero']==i].reset_index(drop=True)
            y = df[df['numero'].isin(range(i, i+tam_janela))]\
                            .reset_index(drop=True).copy()

            if junto:
                y = filtra_sinal_todo(y, high_Wn = high_Wn, high_N = high_N,
                                      low_Wn=low_Wn, low_N = low_N, hann=hann, hamm=hamm,
fltop=fltop, wvlt=wvlt, n_canais=n_canais)
                if n_canais > 1:
                    y['final'] = soma_canais(y, grupo_1, grupo_2, media= False)

                    if y.__len__() < freq_aq*tamanho_janela_segundos and source !=
'tempo_real':
                continue
                elif y.__len__() < 0.9*freq_aq*tamanho_janela_segundos and source ==
'tempo_real':
                continue

            if fft_separada:
                df_new = pd.DataFrame()
                canais = ['canal_'+str(i) for i in range(1, n_canais+1)]
                for ch in canais:
                    # caso um canal esteja ruim no download
                    if y[ch].nunique() < 10 and source == 'download':
                        aux_bool= False
                        continue
                    this = filtra_sinal(y[ch], high_Wn = high_Wn, high_N = high_N,
                                        low_Wn=low_Wn, low_N = low_N, hann=hann, hamm=hamm,
fltop=fltop, wvlt=wvlt)

                                                                        frq,   Y,   t   =
fft(float(this.shape[0])/tamanho_janela_segundos,this)
                    df_new[ch] = Y
                if aux_bool and n_canais != 1:
                      df_new['final'] = soma_canais(df_new, grupo_1, grupo_2, media=
False)
                elif aux_bool:
                    print(df_new.columns)
                    df_new['final'] = df_new['canal_1']
                df_new['frq'] = frq
                y = df_new.copy()
```

```
464.                 #print(amostra)
465.                 #plot_fft(frq,y['final'],frq,y['final'])
466.
467.             if not aux_bool:
468.                 aux_bool=True
469.                 continue
470.             #y = subtrai_media(y)
471.             y = y['final']
472.
473.             if not junto and not fft_separada:
474.                 y = filtra_sinal(y, high_Wn = high_Wn, high_N = high_N,
475.                                  low_Wn=low_Wn, low_N = low_N, hann=hann, hamm=hamm,
    fltop=fltop, wvlt=wvlt)
476.
477.             kappa = pd.DataFrame()
478.             if not fft_separada:
479.                 frq,Y,t=fft(float(y.shape[0])/tamanho_janela_segundos,y)
480.                 kappa['frequencia_olhos'] = frq
481.                 kappa['amplitude'] = Y
482.             else:
483.                 kappa['frequencia_olhos'] = df_new['frq']
484.                 kappa['amplitude'] = df_new['final']
485.             #print(amostra, kappa.sort_values('amplitude',ascending=False).head())
486.             #plot_fft()
487.
488.             # pega as frequencias de maior amplitude na fft do sinal filtrado
489.             kappa = kappa.iloc[0*tamanho_janela_segundos:28*tamanho_janela_segundos]
490.             #print(kappa.shape)
491.             output = get_output_array(amostra, source=source)
492.
493.             if type(dataset) == numpy.ndarray:
494.                                                                     dataset    =
    np.concatenate((dataset,np.array(kappa[['amplitude']]).transpose()), axis=0)
495.                 final_output = np.concatenate((final_output,output), axis=0)
496.             else:
497.                 dataset = np.array(kappa[['amplitude']]).transpose()
498.                 final_output = output
499.
500.                                       #print(kappa[(kappa['frequencia_olhos']<17)   &
    (kappa['frequencia_olhos']>6)]\
501.                                                                                     #
    .sort_values('amplitude',ascending=False).reset_index(drop=True).iloc[0:40])
502.             #print(kappa.sort_values('amplitude', ascending=False))
503.
504.         return dataset, final_output
505.
506.
507.
508.     def roda_processamento(filenames_dic = [], usaveis = [],divisao_minima = 1,
509.                            tamanho_janela_segundos = 4, source = 'tempo_real',
510.                            high_Wn = 0.1, high_N = 8, low_Wn=0.1,
511.                                     low_N = 8, hann=False, hamm=False, fltop=False,
    wvlt=False):
512.         if filenames_dic == []:
513.                                                                 filenames_dic     =
    {'8_0':r'C:\Users\Rafael\Documents\TCC\BCI\EEG_Data\OpenBCI-RAW-teste_8hz.csv',
514.
    '12_0':r'C:\Users\Rafael\Documents\TCC\BCI\EEG_Data\OpenBCI-RAW-teste_12hz.csv',
515.
    '10':r'C:\Users\Rafael\Documents\TCC\BCI\EEG_Data\OpenBCI-RAW-teste_olhos_fechados.
    csv',
```

```
516. 
    '8_1':r'C:\Users\Rafael\Documents\TCC\BCI\EEG_Data\OpenBCI-RAW-20170918_8hz.csv',
517. 
    '15_0':r'C:\Users\Rafael\Documents\TCC\BCI\EEG_Data\OpenBCI-RAW-20170918_15hz.csv',
518. 
    '7':r'C:\Users\Rafael\Documents\TCC\BCI\EEG_Data\OpenBCI-RAW-20170918_7hz.csv',
519.                                                                                '7_1':
    r'C:\Users\Rafael\Documents\TCC\BCI\EEG_Data\OpenBCI-RAW-20170921_7hz.csv',
520. 
    '9':r'C:\Users\Rafael\Documents\TCC\BCI\EEG_Data\OpenBCI-RAW-20170921_9hz.csv',
521. 
    '11':r'C:\Users\Rafael\Documents\TCC\BCI\EEG_Data\OpenBCI-RAW-20170921_11hz.csv',
522. 
    '13':r'C:\Users\Rafael\Documents\TCC\BCI\EEG_Data\OpenBCI-RAW-20170921_13hz.csv',
523. 
    '15_1':r'C:\Users\Rafael\Documents\TCC\BCI\EEG_Data\OpenBCI-RAW-20170921_15hz.csv',
524. 
    '7_2':r'C:\Users\Rafael\Documents\TCC\BCI\EEG_Data\OpenBCI-RAW-20171005_7hz.csv',
525. 
    '7_3':r'C:\Users\Rafael\Documents\TCC\BCI\EEG_Data\OpenBCI-RAW-20171005_7hz_2.csv',
526. 
    '9_2':r'C:\Users\Rafael\Documents\TCC\BCI\EEG_Data\OpenBCI-RAW-20171005_9hz.csv',
527. 
    '9_3':r'C:\Users\Rafael\Documents\TCC\BCI\EEG_Data\OpenBCI-RAW-20171005_9hz_2.csv',
528. 
    '11_2':r'C:\Users\Rafael\Documents\TCC\BCI\EEG_Data\OpenBCI-RAW-20171005_11hz.csv',
529. 
    '11_3':r'C:\Users\Rafael\Documents\TCC\BCI\EEG_Data\OpenBCI-RAW-20171005_11hz_2.csv
    ',
530. 
    '13_2':r'C:\Users\Rafael\Documents\TCC\BCI\EEG_Data\OpenBCI-RAW-20171005_13hz.csv',
531. 
    '13_3':r'C:\Users\Rafael\Documents\TCC\BCI\EEG_Data\OpenBCI-RAW-20171005_13hz_2.csv
    ',
532. 
    '15_2':r'C:\Users\Rafael\Documents\TCC\BCI\EEG_Data\OpenBCI-RAW-20171005_15hz.csv',
533. 
    '15_3':r'C:\Users\Rafael\Documents\TCC\BCI\EEG_Data\OpenBCI-RAW-20171005_15hz_2.csv
    '
534.                         }
535.           usaveis = ['15_0', '7', '7_1', '9', '11', '13', '15_1']
536.             usaveis = [ '7_2', '7_3', '9_2', '9_3', '11_2', '11_3','13_2', '13_3',
    '15_2', '15_3']
537.       elif usaveis ==[]:
538.           usaveis = filenames_dic.keys()
539. 
540.                       #usaveis  = ['9_0',  '9_1',  '11_0',  '11_1',  '13_0',  '13_1',
    '100_0','100_1']
541.           #usaveis = [ '7_2', '7_3','11_2', '11_3',\
542.           #           '13_2', '13_3', '15_2', '15_3']
543. 
544.           #usaveis = [ '7_2', '7_3', '9_2', '9_3', \
545.           #            '13_2', '13_3', '15_2', '15_3']
546.       for index, am in enumerate(usaveis):
547.           amostra = am.split('_')[0]
548.           #print(amostra)
549.                       kap,  output=processa_from_file(filenames_dic[am],  amostra,
    divisao_minima, tamanho_janela_segundos,
550.                                           source=source, high_Wn = high_Wn,
551.                                           high_N = high_N,
552.                                           low_Wn=low_Wn, low_N = low_N,
```

```
553.                                                 hann=hann, hamm=hamm, fltop=fltop,
    wvlt=wvlt)
554.             if index == 0:
555.                 final_dataset = kap
556.                 final_output = output
557.             else:
558.                 final_dataset = (np.concatenate((final_dataset,
559.                                                  kap), axis=0))
560.                 final_output = (np.concatenate((final_output,
561.                                                 output), axis=0))
562.         return final_dataset, final_output
563.
564.     if __name__=="__main__":
565.         pass
566.         size = 100
567.         hidden_layers = 5
568.         hl_tuple = tuple(size for _ in range(hidden_layers))
569.         #final_dataset, final_output = roda_processamento(source = 'not_tempo_real')
570.         #rede = neural_network.rede_neural(final_dataset, final_output, hl_tuple)
571.
572.
573.
574.
```

Arquivo: play_audio.py
Finalidade: Recebe o nome de um arquivo e reproduz o áudio correspondente.

```
1. # -*- coding: utf-8 -*-
2. """
3. play_audio.py
4.
5. @author: Rafael and Lucas
6. """
7.
8. from pygame import mixer # Load the required library
9. import os
10.
11. def play_file(freq):
12.     folder = r'C:\Users\Rafael\Documents\TCC\BCI\EEG_Data\freq_audio'
13.     folder = os.path.join(os.getcwd(), 'freq_audio')
14.     filename = os.path.join(folder, freq+'.mp3')
15.     mixer.init()
16.     #mixer.music.load('e:/LOCAL/Betrayer/Metalik Klinik1-Anak Sekolah.mp3')
17.     mixer.music.load(filename)
18.     mixer.music.play()
19.     while mixer.music.get_busy():
20.         pass
21.     #mixer.music.stop()
22.
```

Arquivo: neural_network.py

Finalidade: Script que recebe os dados processados de EEG, e treina a rede neural. Também usa redes treinadas para prever resultados em tempo real.

```
1. # -*- coding: utf-8 -*-
2. """
3.  neural_network.py
4.
5. @author: Rafael and Lucas
6. """
7.
8. from sklearn.neural_network import MLPClassifier
9. from sklearn.model_selection import train_test_split
10. from sklearn.preprocessing import StandardScaler, scale
11. from sklearn.metrics import classification_report,confusion_matrix
12. import os
13. import random
14. import numpy as np
15.
16. def save_result(conf, classif):
17.     if not os.path.isdir(os.path.join(os.getcwd(),'matrizes')):
18.         os.makedirs(os.path.join(os.getcwd(),'matrizes'))
19.                                                                                 with
    open(os.path.join(os.getcwd(),'matrizes',str(random.randint(1,9999))+'.txt'),   'w')
    as f:
20.         f.write(str(conf) + '\n')
21.         f.write(str(classif))
22.
23. def normalize(dataset):
24.     for index, line in enumerate(dataset):
25.         max_value = line.max()
26.         dataset[index] = line/max_value
27.     return dataset
28.
29. def shuffle(dataset, output):
30.     assert len(dataset) == len(output)
31.     p = np.random.permutation(len(dataset))
32.     return dataset[p], output[p]
33.
34.
35. def rede_neural(final_dataset, final_output, hidden_layer_tuple, split):
36.     #final_dataset=final_dataset[:15]
37.     #final_output=final_output[:15]
38.     #final_dataset = normalize(final_dataset)
39.
40.     # misturando as linhas de entrada e saida
41.     X, y = shuffle(final_dataset, final_output)
42.
43.
44.     if split:
45.         # separando em treinamento e teste
46.         X_train, X_test, y_train, y_test = train_test_split(X, y)
47.     else:
48.         X_train, X_test = X,X
49.         y_train, y_test = y,y
50.
51.
52.     # inicializando o scaler do input
53.     scaler = StandardScaler()
```

```
54.
55.     # fitando a entrada pelo scaler
56.     scaler.fit(X_train)
57.
58.     # agora aplicamos o scaler para o os X
59.     X_train = scaler.transform(X_train)
60.     X_test = scaler.transform(X_test)
61.     #X_train = scale(X_train)
62.     #X_test= scale(X_test)
63.
64.     # incicilizando a rede
65.      rede = MLPClassifier(hidden_layer_sizes=hidden_layer_tuple, max_iter = 9999999,
    solver='lbfgs')
66.
67.     # treinando o modelo
68.     rede.fit(X_train,y_train)
69.
70.     # classificando as linhas de teste
71.     predictions = rede.predict(X_test)
72.
73.     # validação do treinamento
74.     conf_matrix = confusion_matrix(y_test,predictions)
75.     report = classification_report(y_test,predictions)
76.
77.     save_result(conf_matrix, report)
78.
79.     print(conf_matrix)
80.     print(report)
81.     return rede, scaler, get_rate(conf_matrix)
82.
83. def prever(rede, scaler, dataset):
84.     # normalizando a amostra
85.     X = scaler.transform(dataset)
86.
87.     # prevendo a amostra
88.     return rede.predict(X)
89.
90. def prever_debug(rede, scaler, dataset, output):
91.     # normalizando a amostra
92.     X = scaler.transform(dataset)
93.
94.     # prevendo os resultados
95.     predictions = rede.predict(X)
96.
97.     # comparando com os resultados esperados
98.     conf_matrix = confusion_matrix(output,predictions)
99.     report = classification_report(output,predictions)
100.        print(conf_matrix)
101.        print(report)
102.        save_result(conf_matrix, report)
103.        return get_rate(conf_matrix)
104.
105.    def get_rate(matrix):
106.        return float(matrix.trace())/matrix.sum()
```

Arquivo: svm_classifier.py

Finalidade: Script que recebe os dados processados de EEG, e treina um SVM. Também usa objetos treinadas para prever resultados em tempo real.

```
# -*- coding: utf-8 -*-
"""

 svm_classifier.py
@author: Rafael and Lucas
"""

from sklearn import svm
from neural_network import shuffle, get_rate
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, scale
from sklearn.metrics import classification_report,confusion_matrix

import numpy as np

def support_vector_machine(final_dataset, final_output, split):
    # misturando as linhas de entrada e saida
    X, y = shuffle(final_dataset, final_output)

    if split:
        # separando em treinamento e teste
        X_train, X_test, y_train, y_test = train_test_split(X, y)
    else:
        X_train, X_test = X,X
        y_train, y_test = y,y


    # inicializando o scaler do input
    scaler = StandardScaler()

    # fitando a entrada pelo scaler
    scaler.fit(X_train)

    # agora aplicamos o scaler para o os X
    X_train = scaler.transform(X_train)
    X_test = scaler.transform(X_test)
    #X_train = scale(X_train)
    #X_test= scale(X_test)
    # incicilizando a rede

    svm_obj = svm.SVC()

    # treinando o modelo
    svm_obj.fit(X_train,y_train)

    # classificando as linhas de teste
    predictions = svm_obj.predict(X_test)

    conf_matrix = confusion_matrix(y_test,predictions)
    print(conf_matrix)
    print(classification_report(y_test,predictions))
    return svm_obj, scaler, get_rate(conf_matrix)

def prever(svm_obj, scaler, dataset):
    # normalizando a amostra
    X = scaler.transform(dataset)
```

```
57.
58.     # prevendo a amostra
59.     return svm_obj.predict(X)
60.
61. def prever_debug(rede, scaler, dataset, output):
62.     # normalizando a amostra
63.     X = scaler.transform(dataset)
64.
65.     # prevendo os resultados
66.     predictions = rede.predict(X)
67.
68.     conf_matrix = confusion_matrix(output,predictions)
69.     print(conf_matrix)
70.     print(classification_report(output,predictions))
71.     return get_rate(conf_matrix)
```

Arquivo: grid_search.py

Finalidade: Script que recebe uma série de parâmetros de processamento e classificação e executa todas as combinações possíveis, buscando pelo melhor desempenho.

```
# -*- coding: utf-8 -*-

import itertools
import processa_dados_eeg_neural_network_real_time as processa
from multiprocessing import Pool
import pandas as pd
import neural_network
import svm_classifier
import sys
#sys.modules['__main__'].__file__ = 'ipython'

def grid_search(folder, file_dic, file_dic_real, variables_dict, split, nome, tamanho_janela_segundos=4):
    # pegando a posição das variaveis no dicionario
    pos_dict = {}
    try:
        keys = list(variables_dict.keys())
        pos_dict['size'] = keys.index('size')
        pos_dict['hidden_layers'] = keys.index('hidden_layers')
        pos_dict['high_Wn'] = keys.index('high_Wn')
        pos_dict['high_N'] = keys.index('high_N')
        pos_dict['low_Wn'] = keys.index('low_Wn')
        pos_dict['low_N'] = keys.index('low_N')
        pos_dict['hann'] = keys.index('hann')
        pos_dict['hamm'] = keys.index('hamm')
        pos_dict['fltop'] = keys.index('fltop')
        pos_dict['wvlt'] = keys.index('wvlt')
    except:
        print('Pase todas as variávels no dicionário')
        return

    params = list(itertools.product(*variables_dict.values()))
    print('Combinações de parâmetros do grid search: ', len(params))
    p = Pool()

    # iterando nas combinações de param
    print('Executando Grid Search...')

    results = p.map(apply_params, [(file_dic, file_dic_real, param_comb, order,
                                    pos_dict, tamanho_janela_segundos, nome, split)
                                    for order, param_comb in enumerate(params)])
    p.close()
    p.join()

    # concatenado resultados
    final = pd.concat(results, ignore_index=True)

    print('Grid Search finalizado.')
    print(final)
    return final

def apply_params(params):
     file_dic, file_dic_real, param_comb, order, pos_dict, tamanho_janela_segundos,
```

```
    nome, split = params
53.
54.     #print('\n\n\n\n\n\n\n', order,file_dic, '\n\n\n\n\n\n\n')
55.     print('\n\n\n\n\n\n\n', order, '\n\n\n\n\n\n\n')
56.     # seleção do método de classificação
57.     if nome == 'rede_neural':
58.         method = neural_network.rede_neural
59.         module = neural_network
60.     else:
61.         method = svm_classifier.support_vector_machine
62.         module = svm_classifier
63.
64.     # pegando cada param
65.           result  =  {variable:  param_comb[pos_dict[variable]]  for  variable  in
    pos_dict.keys() }
66.
67.     # rodando o processamento com os parametros espeficados
68.        dataset_treino,  output_treino  =  processa.roda_processamento(filenames_dic  =
    file_dic,
69.                                                                    source = 'tempo_real',
    tamanho_janela_segundos=tamanho_janela_segundos,
70.                                                 high_Wn = result['high_Wn'],
71.                                                 high_N = result['high_N'],
72.                                                 low_Wn=result['low_Wn'],
73.                                                 low_N = result['low_N'],
74.                                                 hann=result['hann'],
75.                                                 hamm=result['hamm'],
76.                                                 fltop=result['fltop'],
77.                                                 wvlt=result['wvlt'])
78.
79.     # contruindo o hl_tuple da rede neural
80.     hl_tuple = tuple(result['size'] for _ in range(result['hidden_layers']))
81.     args = tuple((hl_tuple, split))
82.     args = tuple((dataset_treino, output_treino)) + args
83.     rede, scaler, tr_result = method(*args)
84.
85.
86.     # pegando o dataset e o output do aquisição de teste
87.          dataset_real,  output_real  =  processa.roda_processamento(filenames_dic  =
    file_dic_real,
88.                                                                    source = 'tempo_real',
    tamanho_janela_segundos=tamanho_janela_segundos,
89.                                                    high_Wn = result['high_Wn'],
90.                                                    high_N = result['high_N'],
91.                                                    low_Wn=result['low_Wn'],
92.                                                    low_N = result['low_N'],
93.                                                    hann=result['hann'],
94.                                                    hamm=result['hamm'],
95.                                                    fltop=result['fltop'],
96.                                                    wvlt=result['wvlt'])
97.
98.     test_result = module.prever_debug(rede, scaler, dataset_real, output_real)
99.     result['train_result'] = tr_result
100.        result['test_result'] = test_result
101.
102.       return pd.DataFrame([result], columns=result.keys())
```

Arquivo: real_time_controller.py

Finalidade: Script que controla toda a execução do programa. Conecta com a plataforma de aquisição, executa treinamentos e operação em tempo real e chama os scripts de processamento e classificação.

```
# -*- coding: utf-8 -*-
"""

real_time_controller.py
@author: Rafael and Lucas
"""

import os
from datetime import datetime
import processa_dados_eeg_neural_network_real_time as processa
import neural_network
import svm_classifier
import play_audio
import csv
import sys; sys.path.append('..')  # make sure that pylsl is found (note: in a normal program you would bundle pylsl with the program)
import pylsl
import multiprocessing
from multiprocessing.dummy import Pool as ThreadPool
from functools import partial
import inspect
import random
import glob
import re
import pandas as pd
import numpy as np
import pickle
import pca
from grid_search import grid_search

def tempo():
    return float((datetime.utcnow() - datetime(1970, 1, 1)).total_seconds())

def setup():
    # criando a pasta e o nome base dos arquivos
    now = datetime.now()
                                    tm       =       str(now).split()[0]       +'_'+str(now.hour).zfill(2)+'h'+str(now.minute).zfill(2)
    folder = os.path.join(os.getcwd(),'aquisicao_' + tm)
    path_base = os.path.join(folder,tm+'_{}hz.csv')
    if not os.path.isdir(folder):
        os.makedirs(folder)

    # numero de aquisicoes para cada frequencia
    n_aq = 6

    # duracao de cada aquisicao
    duracao = 14
    frequencias = [7,9,11,13, 100] # 100 é repouso
    file_dic = ({key:None for key in
                 [i for j in list(map(lambda x:
                     [str(x)+'_'+str(i) for i in range(n_aq)], frequencias))
                 for i in j]})
```

```
                                                                cabecalho            = 'index_num,canal_1,canal_2,canal_3,canal_4,canal_5,canal_6,canal_7,canal_8,timestamp'
    return path_base, n_aq, duracao, file_dic, cabecalho.split(',')

def abortable_conn(func, *args, **kwargs):
    # função que tentar rodar a connect, e para depois de timeout

    # pegando o timeout da partial que a chama
    timeout = kwargs.get('timeout', None)

    # iniciando a thread que vai rodar a connect
    p = ThreadPool(1)

    # apply_async nessa thread
    res = p.apply_async(func, args=args)
    try:
        # espera timeout segundo para a connect ser completada
        out = res.get(timeout)
        return out
    except multiprocessing.TimeoutError:
        # caso dê timeout, levanta um erro
        print("Não foi possível estabeler conexão com o OpenBCI!")
        p.terminate()
        raise

def connect_controller():
      # função que controla a chamada da abortable_conn, que por sua vez chama a connect
    # quando for conectar ao openbci, chamar ESTA FUNÇÃO
    try:
        # instancia abortable_conn(func = connect, kwargs = timeout)
        abortable_func = partial(abortable_conn, connect, timeout=10)

        #rodando a função
        conn = abortable_func()
        return conn
    except:
        return

def connect():
    # função que efetivamente conecta com o open BCI
    if inspect.stack()[1][3] != 'worker':
                    print('ATENÇÃO!  Não  chamar  o  connect  diretamente!  Usar  o connect_controller')
        return

    #  first resolve an EEG stream on the lab network
    print("Procurando pela stream EEG...")
    streams = pylsl.resolve_stream('type','EEG')

    # create a new inlet to read from the stream
        inlet = pylsl.stream_inlet(streams[0])

        #print(timestamp, list(sample))
        while True:
            tm = inlet.pull_sample([])
            if tm[0]:
                break
        return streams

```

```
109.   def training_aquisition(streams, divisao_minima = 1):
110.       # pegando os parâmetros de teste
111.       path_base, naq, duracao, file_dic, cabecalho = setup()
112.       #inlet, sample = connect_controller()
113.
114.       end = False
115.       # preenche todos os arquivos de treinamento
116.       while not end:
117.           # pega a lista das aquisições que ainda não foram
118.           available = list(filter(lambda x: file_dic[x] is None, file_dic.keys()))
119.
120.           # se todas tiverem sido preenchidas, sai
121.           if not available:
122.               end=True
123.               break
124.
125.           # sorteia uma frequencia
126.           current = random.choice(available)
127.           filename = path_base.format(current)
128.
129.           # abre o arquivo de saida
130.           f = open(filename, 'w')
131.           csv_writer = csv.writer(f, lineterminator = '\n')
132.           csv_writer.writerow(cabecalho)
133.
134.           # avisa o usuario para olhar para uma determinada luz
135.           print('\n\n\nOlhe para o led de %s Hertz.' %(current.split('_')[0]))
136.           play_audio.play_file(current.split('_')[0])
137.
138.           # definindo o inlet
139.           inlet = pylsl.stream_inlet(streams[0])
140.
141.           # aquisitando
142.
143.           start = tempo()
144.           last_it = tempo()
145.           index = 0
146.           while tempo() - start <=  duracao:
147.               sample = []
148.               it_time = tempo()
149.               if it_time - last_it >= divisao_minima:
150.                   last_it = tempo()
151.                   index = 0
152.               timestamp = inlet.pull_sample(sample)
153.               if timestamp[0]:
154.                   index += 1
155.                   csv_writer.writerow([index]+timestamp[0]+[timestamp[1]])
156.
157.           print('Pode parar de olhar.')
158.
159.           # fechando o arquivo
160.           f.close()
161.           file_dic[current] = filename
162.
163.       play_audio.play_file('Fim')
164.       return file_dic, path_base
165.
166.   def get_file_dic(folder = '', keyword='aquisicao', dataset = 0 ):
167.       if not folder:
168.           folders = glob.glob(os.path.join(os.getcwd(), keyword + '*'))
169.           folders.sort()
```

```
170.           folder = folders[-1]
171.       print('Usando os arquivos da pasta ', folder)
172.       """2017-11-05_17h35 ficou muito bom"""
173.       files = glob.glob(folder+r'\*hz.csv')
174.       if keyword:
175.              file_dic = {re.findall('h[0-9]{2}_(.*?)hz',file)[0]:file for file in
    files}
176.       else:
177.           file_dic = {re.findall('h(.*?)hz',file)[0]:file for file in files}
178.       return file_dic
179.
180.               def  run_training(streams,  method,  args,  divisao_minima  =  1,
    tamanho_janela_segundos = 4,
181.                     high_Wn = 0.1, high_N = 8, low_Wn=0.1,
182.                     low_N = 8, hann=False, hamm=False, fltop=False, wvlt=False):
183.
184.       # chamando a aquisição dos arquivos
185.       file_dic, path_base = training_aquisition(streams)
186.
187.       # processando os sinais
188.       dataset, output = processa.roda_processamento(
189.               filenames_dic = file_dic,
190.               divisao_minima = divisao_minima,
191.               tamanho_janela_segundos = tamanho_janela_segundos,
192.               high_Wn = high_Wn, high_N = high_N,
193.                    low_Wn=low_Wn,low_N = low_N, hann=hann, hamm=hamm, fltop=fltop,
    wvlt=wvlt)
194.
195.       args = tuple((dataset, output)) + args
196.       rede, scaler, result = method(*args)
197.       print(file_dic)
198.       return rede, scaler, path_base, file_dic
199.
200.   def passa_lista(lista, element, size):
201.       if len(lista) == size:
202.           lista_return = [lista[i+1] for i in range(0, size-1)] + [element]
203.       else:
204.           lista_return = lista + [element]
205.       return lista_return
206.
207.   def run_debug_mode(rede, scaler, module, streams, path,
208.                               divisao_minima = 1, tamanho_janela_segundos = 4,
209.                               size = 10, full=False,
210.                               high_Wn = 0.1, high_N = 8, low_Wn=0.1,
211.                                    low_N = 8, hann=False, hamm=False, fltop=False,
    wvlt=False):
212.       possible_freqs = {'7':0, '9':1, '11':2, '13':3, '100':5}
213.       i = 0
214.
215.       duration = 30
216.       n = 1
217.       aq=0
218.
219.       print(path)
220.
221.       # vetor para guardar o sinal do open bci
222.       signal = []
223.
224.       # index da posição de cada sinal na janela
225.       index = 0
226.
```

```
227.       # lista que guardará os indexes maximos das janelas anteriores
228.       index_lista = []
229.
230.       # quantos outputs são guardados
231.       output_size = 100
232.       # lista para guardar esses outputs
233.       output_lista = []
234.
235.       # cabecalho do dataframe
236.                                                                   cabecalho      =
    'index_num,canal_1,canal_2,canal_3,canal_4,canal_5,canal_6,canal_7,canal_8,timestam
    p,freq,aq'
237.       cols = cabecalho.split(',')
238.
239.       # tempo no inicio da aquisição
240.       inicio = tempo()
241.       last_time = tempo()
242.       current_time = tempo()
243.
244.       if full:
245.           # definindo o inlet
246.           inlet = pylsl.stream_inlet(streams[0])
247.
248.           # dataframe para guardar as os de varias divisoes
249.           df = pd.DataFrame(columns = cols)
250.           tam_divisao = int(tamanho_janela_segundos/divisao_minima)
251.
252.           # file e objetos para salvar o sinal
253.           filename = os.path.join(path, 'tempo_real.csv')
254.           file_obj = open(filename, 'w')
255.           writer = csv.writer(file_obj, lineterminator = '\n')
256.           writer.writerow(cols)
257.
258.           # escolhendo uma label
259.           freq, i = choose_freq_play_file(possible_freqs, i)
260.           # definindo o tempo para duracao total
261.           tempo_0 = tempo()
262.           duracao_total = duration*(n+1)*(len(possible_freqs.keys()))
263.
264.
265.           while True:
266.               sample = []
267.               # caso passe a divisão minima, pega os resultados, processa e preve
268.               if current_time - last_time >= divisao_minima:
269.
270.                   # passa_lista, para salvar o index dessa iteração
271.                   index_lista = passa_lista(index_lista, index, size)
272.
273.                   # reseta o valor index (controle)
274.                   index = 0
275.
276.                   # o tempo será tirado a partir de agora
277.                   last_time = current_time
278.
279.                   # do dataframe que guarda uma janela, tira a mais antiga
280.                   if len(index_lista) > tam_divisao:
281.                                                                           df   =
    df.iloc[index_lista[-tam_divisao-1]:].copy().reset_index(drop=True)
282.
283.                   # e poe a nova
284.                   df = pd.concat((df,pd.DataFrame(signal, columns = cols)))
```

```
285.
286.                     # reseta o vetor de entrada
287.                     signal = []
288.                     # processa o sinal
289.                     if len(index_lista) > tam_divisao:
290.                         sinal_processado = processa.processa_real_time(df.copy(),
291.
    tamanho_janela_segundos = tamanho_janela_segundos,
292.                                                                        high_Wn =
    high_Wn,
293.                                                                         high_N =
    high_N,
294.
    low_Wn=low_Wn, low_N = low_N,
295.                                                                     hann=hann,
    hamm=hamm, fltop=fltop, wvlt=wvlt)
296.
297.                         # transforma o output do processamento em um np array
298.                         dataset = np.array(sinal_processado)
299.
300.                         # tenta prever o estado
301.                         saida = module.prever(rede, scaler, dataset)
302.
303.                         print(saida)
304.
305.                         # salva o output
306.                         output_lista = passa_lista(output_lista, saida, output_size)
307.
308.                 timestamp = inlet.pull_sample(sample)
309.                 if timestamp[0]:
310.                     current_time = tempo()
311.                     index = index + 1
312.
313.                     # append na lista
314.                     signal.append([index]+timestamp[0]+[timestamp[1], freq, aq])
315.
316.                     # salva o sinal
317.                     writer.writerow([index]+timestamp[0]+[timestamp[1], freq, aq])
318.
319.                 if current_time-inicio >= duration:
320.                     inicio = tempo()
321.                     freq,i = choose_freq_play_file(possible_freqs, i)
322.                     aq+=1
323.                     # redefinindo o inlet
324.                     inlet = pylsl.stream_inlet(streams[0])
325.
326.                 if current_time - tempo_0 >= duracao_total:
327.                     play_audio.play_file('Fim')
328.                     break
329.             file_obj.close()
330.         file_dic = transform_real_time_files(path, possible_freqs)
331.         #file_dic = get_file_dic('')
332.         dataset, output = processa.roda_processamento(filenames_dic = file_dic,
333.                                                                 source = 'tempo_real',
    tamanho_janela_segundos=tamanho_janela_segundos,
334.                                                       high_Wn = high_Wn,
335.                                                       high_N = high_N,
336.                                                             low_Wn=low_Wn, low_N =
    low_N,
337.                                                              hann=hann, hamm=hamm,
    fltop=fltop, wvlt=wvlt)
```

```
338.
339.         module.prever_debug(rede, scaler, dataset, output)
340.         return file_dic
341.
342.     def choose_freq_play_file(possible_freqs, i):
343.         freq = list(possible_freqs.keys())[i]
344.         if i == len(possible_freqs)-1:
345.             i=0
346.         else:
347.             i+=1
348.         play_audio.play_file(freq)
349.         return freq, i
350.
351.     def transform_real_time_files(folder, possible_freqs):
352.         arquivo = os.path.join(folder, 'tempo_real.csv')
353.         freq_counts = {key:0 for key in possible_freqs.keys()}
354.         df = pd.read_csv(arquivo)
355.         file_dic = {}
356.         if not os.path.isdir(os.path.join(folder,'debug')):
357.             os.makedirs(os.path.join(folder,'debug'))
358.         for i in df['aq'].unique():
359.             temp = df[df['aq']== i].copy()
360.             if temp.shape[0] < 2500:
361.                 continue
362.             freq = str(temp['freq'].unique()[0])
363.             assert temp['freq'].nunique() == 1
364.             file_label = str(freq)+'_'+str(freq_counts[freq])
365.             freq_counts[freq] +=1
366.             filename = os.path.join(folder,'debug', file_label+'hz.csv')
367.             file_dic[file_label] = filename
368.             temp.to_csv(filename, index=False)
369.         return file_dic
370.
371.
372.         def run_forever(rede, scaler, module, streams, path, divisao_minima = 1,
    tamanho_janela_segundos = 4, size = 10,
373.                         high_Wn = 0.1, high_N = 8, low_Wn=0.1,
374.                         low_N = 8, hann=False, hamm=False, fltop=False, wvlt=False):
375.         # vetor para guardar o sinal do open bci
376.         signal = []
377.
378.         # index da posição de cada sinal na janela
379.         index = 0
380.
381.         # lista que guardará os indexes maximos das janelas anteriores
382.         index_lista = []
383.
384.         # quantos outputs são guardados
385.         output_size = 100
386.         # lista para guardar esses outputs
387.         output_lista = []
388.
389.         # cabecalho do dataframe
390.                                                                                 cabecalho      =
    'index_num,canal_1,canal_2,canal_3,canal_4,canal_5,canal_6,canal_7,canal_8,timestam
    p'
391.         cols = cabecalho.split(',')
392.
393.         # dataframe para guardar as os de varias divisoes
394.         df = pd.DataFrame(columns = cols)
395.         tam_divisao = int(tamanho_janela_segundos/divisao_minima)
```

```
396.
397.       # file e objetos para salvar o sinal
398.       filename = os.path.join(path, 'tempo_real.csv')
399.       file_obj = open(filename, 'w')
400.       writer = csv.writer(file_obj, lineterminator = '\n')
401.       writer.writerow(cols)
402.
403.       # tempo no inicio da aquisição
404.
405.       last_time = tempo()
406.       current_time = tempo()
407.
408.       # definindo o inlet
409.       inlet = pylsl.stream_inlet(streams[0])
410.
411.       while True:
412.           sample = []
413.           # caso passe a divisão minima, pega os resultados, processa e preve
414.           if current_time - last_time >= divisao_minima:
415.               # passa_lista, para salvar o index dessa iteração
416.               index_lista = passa_lista(index_lista, index, size)
417.
418.               # reseta o valor index (controle)
419.               index = 0
420.
421.               # o tempo será tirado a partir de agora
422.               last_time = current_time
423.
424.               # do dataframe que guarda uma janela, tira a mais antiga
425.               if len(index_lista) > tam_divisao:
426.                                                                                   df    =
    df.iloc[index_lista[-tam_divisao-1]:].copy().reset_index(drop=True)
427.
428.               # e poe a nova
429.               df = pd.concat((df,pd.DataFrame(signal, columns = cols)))
430.
431.               # reseta o vetor de entrada
432.               signal = []
433.               # processa o sinal
434.               if len(index_lista) > tam_divisao:
435.                   sinal_processado = processa.processa_real_time(df.copy(),
436.
    tamanho_janela_segundos = tamanho_janela_segundos,
437.                                                                              high_Wn =
    high_Wn,
438.                                                                               high_N =
    high_N,
439.
    low_Wn=low_Wn, low_N = low_N,
440.                                                                           hann=hann,
    hamm=hamm, fltop=fltop, wvlt=wvlt)
441.
442.                   # transforma o output do processamento em um np array
443.                   dataset = np.array(sinal_processado)
444.
445.                   # tenta prever o estado
446.                   saida = module.prever(rede, scaler, dataset)
447.
448.                   print(saida)
449.
450.                   # salva o output
```

```
451.                     output_lista = passa_lista(output_lista, saida, output_size)
452.
453.             timestamp = inlet.pull_sample(sample)
454.             if timestamp[0]:
455.                 current_time = tempo()
456.                 index = index + 1
457.
458.
459.                 # append na lista
460.                 signal.append([index]+timestamp[0]+[timestamp[1]])
461.
462.                 # salva o sinal
463.                 writer.writerow([index]+timestamp[0]+[timestamp[1]])
464.
465.
466.     def salvar_objeto(objeto, folder, nome=''):
467.         if not nome:
468.             print('Forneça o nome do objeto')
469.             return
470.         print('Salvando objeto em ',os.path.join(folder, nome + '.pkl'))
471.         with open(os.path.join(folder, nome + '.pkl'), 'wb') as f:
472.             pickle.dump(objeto, f)
473.
474.     def carregar_objeto(path):
475.         with open(path, 'rb') as f:
476.             obj = pickle.load(f)
477.         return obj
478.
479.     def run(treinar = True, adquirir = False, rede_path = '', file_dic = {},
480.             rede = '', divisao_minima = 1, tamanho_janela_segundos = 4,
481.                      size = 80, hidden_layers = 3, debug=True, full_debug=False,
    search=False,
482.             open_bci=False, split = True, high_Wn = 0, high_N = 6, low_Wn=0.84,
483.             low_N = 2, hann=False, hamm=True, fltop=False, wvlt=False):
484.
485.         # seleção do método de classificação
486.                                             methods    =    [neural_network.rede_neural,
    svm_classifier.support_vector_machine]
487.         method = methods[0]
488.
489.         if method == neural_network.rede_neural:
490.             print('Método de classificação escolhido: Rede Neural')
491.             module = neural_network
492.             hl_tuple = tuple(size for _ in range(hidden_layers+1))
493.             args = tuple((hl_tuple, split))
494.             nome = 'rede_neural'
495.         else:
496.             print('Método de classificação escolhido: SVM')
497.             module = svm_classifier
498.             args = tuple((split,))
499.             nome = 'svm'
500.
501.
502.         # tentando conectar ao OpenBCI
503.         if open_bci:
504.             try:
505.                 streams = connect_controller()
506.                 if not streams:
507.                     return
508.             except:
509.                 return
```

```
510.        else:
511.            streams = [1]
512.
513.        # definindo os modos de operação
514.        if rede_path and rede:
515.            print('Você entrou com um path e uma rede, entre com apenas um')
516.            return
517.
518.        elif treinar and adquirir:
519.                rede, scaler, path, file_dic = run_training(streams, method, args,
    tamanho_janela_segundos=4, divisao_minima=1,
520.                                                            high_Wn = high_Wn,
521.                                                            high_N = high_N,
522.                                                            low_Wn=low_Wn,low_N = low_N,
523.                                                                   hann=hann, hamm=hamm,
    fltop=fltop, wvlt=wvlt)
524.            folder = os.path.dirname(path)
525.            salvar_objeto(rede, folder, nome='rede_neural')
526.            salvar_objeto(scaler, folder, nome='scaler')
527.
528.        elif treinar and not adquirir and not rede_path:
529.                                                                    folder     =
    'C:\\Users\\Rafael\\Documents\\TCC\\BCI\\EEG_Data\\aquisicao_2017-11-07_10h30'
530.                                                                    folder     =
    r'C:\Users\Rafael\Documents\TCC\BCI\EEG_Data\aquisicao_2017-11-10_10h11'
531.                                                                    folder     =
    r'C:\Users\Rafael\Documents\TCC\BCI\EEG_Data\aquisicao_2017-11-11_16h14'
532.                                                                    folder     =
    r'C:\Users\Rafael\Documents\TCC\BCI\EEG_Data\aquisicao_2017-11-09_11h29'
533.                                                                   #folder     =
    r'C:\Users\Rafael\Documents\TCC\BCI\EEG_Data\aquisicao_2017-11-13_14h19'
534.                                                                   #folder     =
    'C:\\Users\\Rafael\\Documents\\TCC\\BCI\\EEG_Data\\aquisicao_2017-11-11_17h01'
535.                                                                   #folder     =
    'C:\\Users\\Rafael\\Documents\\TCC\\BCI\\EEG_Data\\aquisicao_2017-11-12_22h43'     #
    nelson
536.            #folder = ''
537.
538.            if not file_dic:
539.                file_dic = get_file_dic(folder)
540.            folder = os.path.dirname(list(file_dic.values())[0])
541.
542.            print('Processando dados...')
543.            dataset, output = processa.roda_processamento(filenames_dic = file_dic,
544.                                                                  source = 'tempo_real',
    tamanho_janela_segundos=tamanho_janela_segundos,
545.                                                           high_Wn = high_Wn,
546.                                                           high_N = high_N,
547.                                                               low_Wn=low_Wn, low_N =
    low_N,
548.                                                                hann=hann, hamm=hamm,
    fltop=fltop, wvlt=wvlt)
549.            #pca.plot_pca(dataset, output)
550.            args = tuple((dataset, output)) + args
551.
552.            # treinando o ML
553.            rede, scaler, result = method(*args)
554.
555.            # salvando o objeto
556.            salvar_objeto(rede, folder, nome = nome)
557.            salvar_objeto(scaler, folder, nome='scaler')
```

```
558.
559.       elif not treinar and not rede_path and not rede:
560.               print('Você escolheu entrar com uma rede treinada, mas não forneceu
    uma!')
561.           return
562.       elif (not treinar or not adquirir) and rede_path:
563.           path = setup()[0]
564.           folder = os.path.dirname(path)
565.           print(folder)
566.           try:
567.               rede = carregar_objeto(rede_path)
568.               scaler = carregar_objeto(rede_path.replace('rede_neural', 'scaler'))
569.           except:
570.               print('Caminho para a rede não encontrado!')
571.               return
572.       elif not treinar and rede:
573.           pass
574.
575.       if debug:
576.             file_dic_real = run_debug_mode(rede, scaler, module, streams, folder,
    divisao_minima,
577.                                       tamanho_janela_segundos, full=full_debug,
578.                                       high_Wn = high_Wn,
579.                                       high_N = high_N,
580.                                       low_Wn=low_Wn, low_N = low_N,
581.                                       hann=hann, hamm=hamm, fltop=fltop, wvlt=wvlt)
582.       elif not debug and not search:
583.           print('Iniciando operação em tempo real...')
584.           print('Boa Sorte!')
585.           run_forever(rede, scaler, module, streams, folder, divisao_minima,
586.                                       tamanho_janela_segundos,
587.                                       high_Wn = high_Wn,
588.                                       high_N = high_N,
589.                                       low_Wn=low_Wn,low_N = low_N,
590.                                       hann=hann, hamm=hamm, fltop=fltop, wvlt=wvlt)
591.       if debug and search:
592.           variables_dict = {
593.               'size': [50,100],
594.               'hidden_layers':[3,5],
595.               'high_Wn':[False,0.1,0.25,0.4],
596.               'high_N':[8,10],
597.               'low_Wn':[False,0.25,0.4,0.55,0.7,0.8,0.9],
598.               'low_N':[6,8],
599.               'hann':[True,False],
600.               'hamm':[True,False],
601.               'fltop':[False],
602.               'wvlt':[True,False]
603.
604.           }
605.             results = grid_search(folder, file_dic, file_dic_real, variables_dict,
    split,
606.                                   nome, tamanho_janela_segundos=tamanho_janela_segundos)
607.           results.to_csv(os.path.join(folder, 'results.csv'), index=False)
608.
609.           """
610.                   run(adquirir=False, debug=True, full_debug=False, search=True,
    open_bci=False)
611.           """
612.
613.   def run_datasets(user, dataset=2, divisao_minima = 1, tamanho_janela_segundos =
    4,
```

```
614.                         size = 40, hidden_layers = 3, debug=True, search=False,
615.                         split = True, high_Wn = 0, high_N = 4, low_Wn=0,
616.                         low_N = 5, hann=False, hamm=False, fltop=False, wvlt=False):
617.
618.       # seleção do método de classificação
619.                                         methods    =    [neural_network.rede_neural,
    svm_classifier.support_vector_machine]
620.       method = methods[0]
621.
622.       if method == neural_network.rede_neural:
623.           print('Método de classificação escolhido: Rede Neural')
624.           module = neural_network
625.           hl_tuple = tuple(size for _ in range(hidden_layers+1))
626.           args = tuple((hl_tuple, split))
627.           nome = 'rede_neural'
628.       else:
629.           print('Método de classificação escolhido: SVM')
630.           module = svm_classifier
631.           args = tuple((split,))
632.           nome = 'svm'
633.
634.       # pegando o usuario e a pasta correspondente
635.       user = str(user).zfill(3)
636.       if dataset == 1:
637.           folder = os.path.join(os.getcwd(), 'download_eeg_data', user)
638.       elif dataset == 2:
639.           folder = os.path.join(os.getcwd(), 'download_eeg_data_2', 'data', user)
640.       file_dic = get_file_dic(folder, keyword='', dataset = dataset)
641.       folder = os.path.dirname(list(file_dic.values())[0])
642.
643.       print('Processando dados...')
644.       dataset, output = processa.roda_processamento(filenames_dic = file_dic,
645.                                                     source = 'download',
    tamanho_janela_segundos=tamanho_janela_segundos,
646.                                                 high_Wn = high_Wn,
647.                                                 high_N = high_N,
648.                                                 low_Wn=low_Wn, low_N = low_N,
649.                                                         hann=hann, hamm=hamm,
    fltop=fltop, wvlt=wvlt)
650.
651.       print('Dados Processados. Treinando a rede neural...')
652.       #pca.plot_pca(dataset, output)
653.       args = tuple((dataset, output)) + args
654.
655.       # treinando o ML
656.       rede, scaler, result = method(*args)
657.
658.       # salvando o objeto
659.       salvar_objeto(rede, folder, nome = nome)
660.       salvar_objeto(scaler, folder, nome='scaler')
661.       print('FIM')
662.
663.
664.   if __name__ == "__main__":
665.                   #run(adquirir=False,  debug=True,  full_debug=False,  search=True,
    open_bci=False)
666.       #run()
667.       #for i in range(1):
668.       #    run_datasets(3)
669.       run()
670.       pass
```

ANEXO A: ESPECIFICAÇÕES TÉCNICAS DA PLATAFORMA CYTON (OPENBCI)

OpenBCI Cyton

The OpenBCI Cyton PCBs were designed with Design Spark, a free PCB capture program. You can find a link to download Design Spark in our V3 design files repository where you will find all of the .sch and .pcb files. There are parts in the BOMs below that are not explicitly specified. For example, the passives (Rs and Cs) are all standard easy to find components. Thin film for the Rs, and MLCC X7R for the Cs. The battery connector is a standard JST type two position (with polarity key at the **TOP**), and the SD card holder that we are using is ST-TF-003A.

OpenBCI Cyton Board

Cyton Board Specs:

- Power with 3-6V DC Battery ONLY
- PIC32MX250F128B Mircontroller with chipKIT UDB32-MX2-DIP bootloader
- ADS1299 Analog Front End
- LIS3DH 3 axis Accelerometer
- RFduino BLE radio
- Micro SD card slot
- Voltage Regulation (3V3, +2.5V, -2.5V)
- Board Dimensions 2.41” x 2.41” (octogon has 1” edges)
- Mount holes are 1/16” ID, 0.8” x 2.166” on center

Breakout pins:

- Program pins for bootloading PIC
    - PGC, PGD, VDD, MCLR, GND
- Serial pins for programming RFduino
    - RFTX, RFRX, RFRST, GND
- SPI bus pins on the 3V side for Daisy Module expansion
    - DVDD, GND, MISO, MOSI, SCK, CS, CLK, RST
- Unused PIC32 pins
    - D11 (A5), D12 (A6), D13 (A7), D17, D18

The SPI bus pins on 3V side include CLK, which is tied to the ADS1299 CLK pin for timing the Daisy Module, and a RST pin which is tied to the ADS1299 MCLR pin for hardware reset of the ADS chips. We use a PICkit 3 to bootload the PIC chips. Pins D11, D12, and D13 can be digital or analog (called by their A number above for analog purposes). D11 is also PGD, and has the blue LED in series with a 1K resistor connected to AGND. D12 is PGC, for bootloading purposes. D17 and D18 are digital I|O only. D17 is connected to the PROG pushbutton. The PROG button can be used as an input by setting it’s MODE direction and doing digitalRead on it (there is a 470K pulldown on D17, pressing PROG pulls pin D17 up to DVDD).

Push Buttons

The RST pushbutton is connected to MCLR on the PIC. Pressing it will reset the PIC. To put the PIC into bootloader mode so that it can be re-programmed, press the RST button and hold it down. Then press the PROG button. Then release the RST button, and the blue LED will blink pleasantly, announcing that the PIC is ready to accept new code.

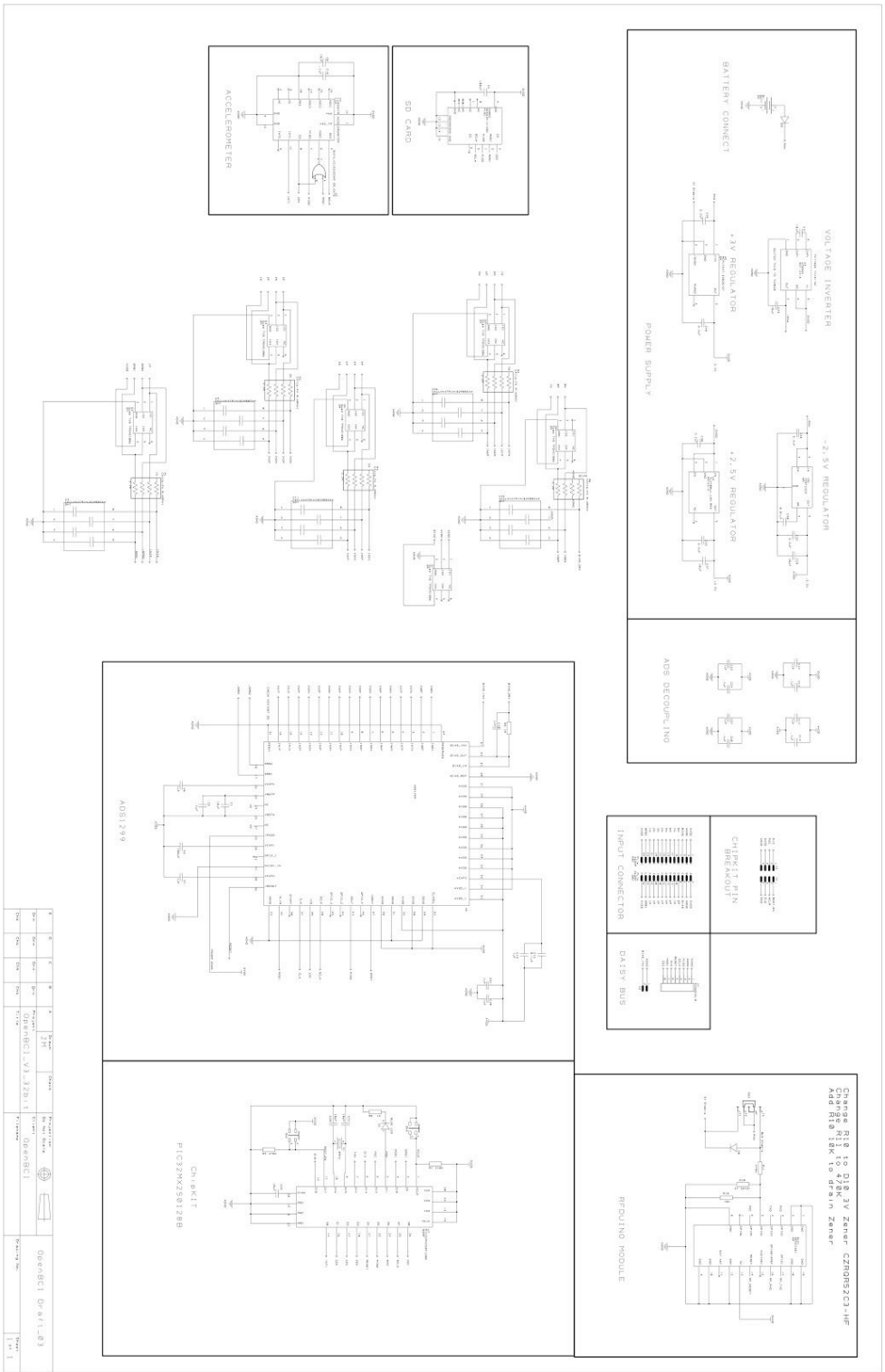
Slide Switch

Slide switch is power for the board. The slide switch has three positions:

- BLE activates a pull-up on RFduino pin 4
- OFF disconnects the battery input
- PC does NOT activate pull-up on RFduino pin 4
- **NOTE: BLE|PC selection is NOT implemented!**

Swtiching either BLE or PC will produce the same result. The option is available for future development…

OpenBCI Cyton Board Circuit Schematic



OpenBCI USB DONGLE

The OpenBCI USB Dongle is used to connect your computer to the Cyton Board.

Dongle Specs

- Power via USB connector ONLY
- RFduino BLE radio module
- FTDI USB<>Serial IC (FT231XQ-R)
- Resettable fuse

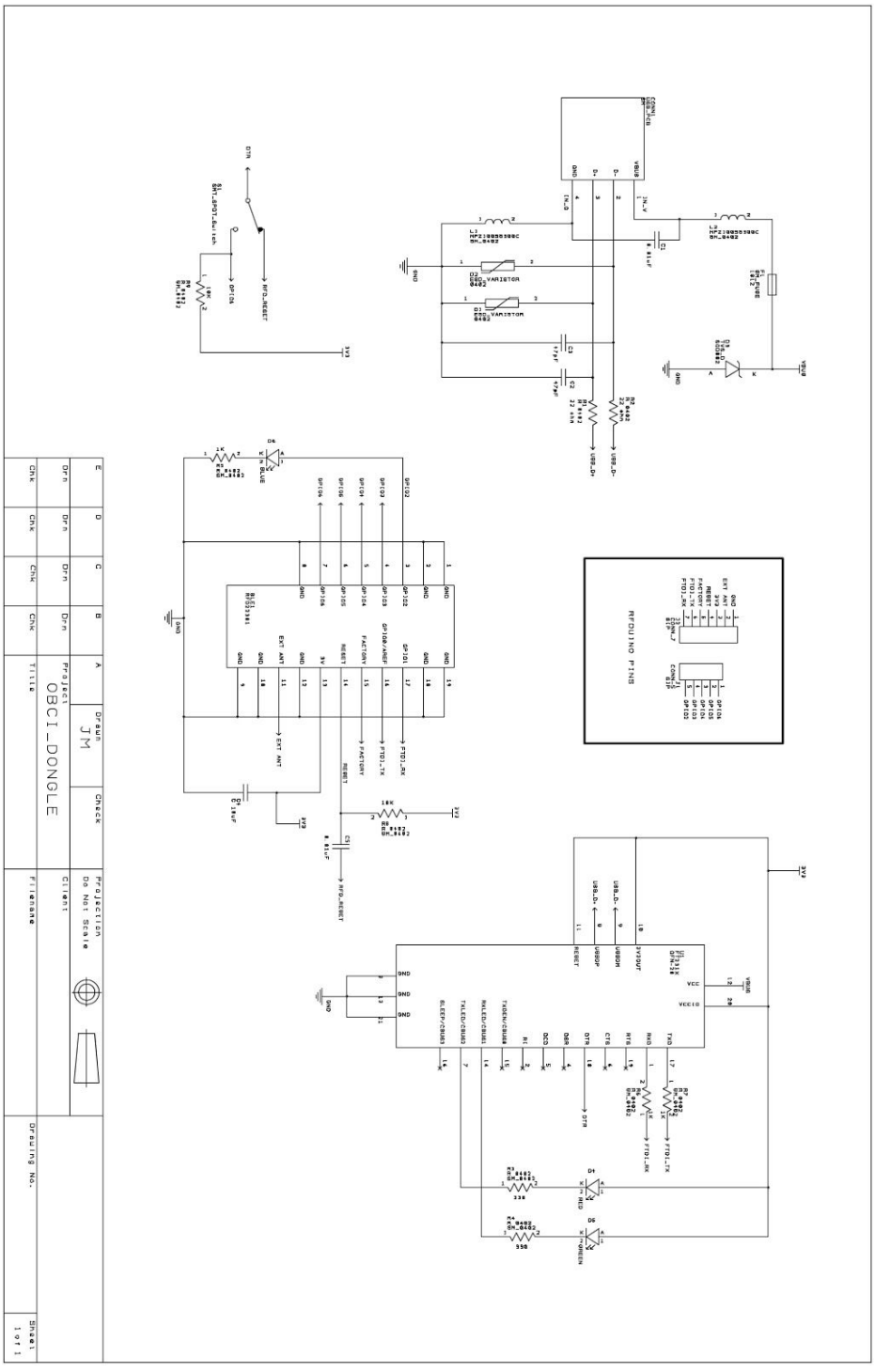
Breakout Pins

RFduio pins are broken out in the same order and layout as the RFduino radio and shields. That makes the OpenBCI USB Dongle compatible with the RFduino shields, if you like. The TXD (red) and RXD (greed) LEDs are connected to outputs from the FTDI chip. The blue LED is connected to RFduino GPIO2.

Slide Switch

The slide switch on the Dongle has two positions (noted on the bottom silkscreen). When the switch is on the GPIO6 side, the FTDI DTR pin is routed to RFduino pin 6 and it is ready to pass data to-from the Cyton board. This configuration is ‘normal’ mode, and also allows for programming the Cyton board over air. When the switch is on the RESET side, the FTDI DTR pin is routed to the RFduino RESET pin. This mode allows for re-programming the RFduino on the Dongle.

OpenBCI Dongle Circuit Schematic



ANEXO B: ESPECIFICAÇÕES TÉCNICAS DA PLATAFORMA TMSI REFA

Refa

Technical Specifications

REVISIÓN 4

TMSi

2017

95-0108-324-3, Refa8-24e 2048Hz

Type	**Refa8-24e**
REF code	95-0108-324-3

Unipolar ExG inputs (EEG, ECG, EOG, EMG etc):

Number	24
RMS Noise	< 1 µV (@ lowest sample frequency)
Gain	20 x
Input signal difference	-150 mV to +150 mV (@ 0 V common signal)
Input common mode range	-2 V to +2 V (@ 0 V differential signal)
Input impedance	> 100 MΩ
CMRR	> 90 dB
Connector	safety din per channel and subD37 female connector per 32 channels

Digital input

Input turn-on current	2 mA @ 3 V input, max. input = 5 V
Isolation	> 4000 V, by means of optocoupler (H11L1)
Connector	8 bit via DB25 female, shared first bit via BNC female

Sampling:

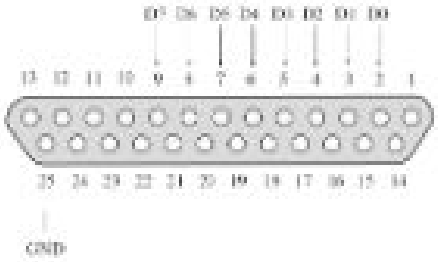
Number of channels	24 channels simultaneously
Resolution	22 bits, ExG 0.0715 µV per bit
Sample frequency	2048 Hz, 1024 Hz, 512 Hz, 256 Hz, 128 Hz

Channel list:

nr	name	Function		resolution	range
1	A1	Unipolar input 1		0.0715 µV	-150 mV to +150 mV
2	C3	Unipolar input 2		0.0715 µV	-150 mV to +150 mV
3	P3	Unipolar input 3		0.0715 µV	-150 mV to +150 mV
4	Cz	Unipolar input 4		0.0715 µV	-150 mV to +150 mV
5	Oz	Unipolar input 5		0.0715 µV	-150 mV to +150 mV
6	C4	Unipolar input 6		0.0715 µV	-150 mV to +150 mV
7	T6	Unipolar input 7		0.0715 µV	-150 mV to +150 mV
8	A2	Unipolar input 8		0.0715 µV	-150 mV to +150 mV
9	T3	Unipolar input 9		0.0715 µV	-150 mV to +150 mV
10	T5	Unipolar input 10		0.0715 µV	-150 mV to +150 mV
11	O1	Unipolar input 11		0.0715 µV	-150 mV to +150 mV
12	Pz	Unipolar input 12		0.0715 µV	-150 mV to +150 mV
13	O2	Unipolar input 13		0.0715 µV	-150 mV to +150 mV
14	P4	Unipolar input 14		0.0715 µV	-150 mV to +150 mV
15	F8	Unipolar input 15		0.0715 µV	-150 mV to +150 mV
16	T4	Unipolar input 16		0.0715 µV	-150 mV to +150 mV
17	F7	Unipolar input 17		0.0715 µV	-150 mV to +150 mV
18	Fp1	Unipolar input 18		0.0715 µV	-150 mV to +150 mV
19	F3	Unipolar input 19		0.0715 µV	-150 mV to +150 mV
20	Fz	Unipolar input 20		0.0715 µV	-150 mV to +150 mV
21	Fpz	Unipolar input 21		0.0715 µV	-150 mV to +150 mV
22	Fp2	Unipolar input 22		0.0715 µV	-150 mV to +150 mV
23	F4	Unipolar input 23		0.0715 µV	-150 mV to +150 mV
24	XD	Unipolar input 24		0.0715 µV	-150 mV to +150 mV
25	Digi	Digital channel (bits)		1 (bit)	
		0	Digital input bit 0		
		1	Digital input bit 1		
		2	Digital input bit 2		
		3	Digital input bit 3		
		4	Digital input bit 4		
		5	Digital input bit 5		
		6	Digital input bit 6		
		7	Digital input bit 7 (MSB)		
		8	always 1		
		9-13	Sawtooth test signal		
		14-31	reserved		

Digital input DB25 connector

Pin	Input
2	bit 0 (parallel to BNC connector in software)
3	bit 1
4	bit 2
5	bit 3
6	bit 4
7	bit 5
8	bit 6
9	bit 7 (MSB)
25	common ground



Headcap connector

This table describes the relation between signal channel numbers and headcap connector pin numbers.

DB37 pin number	Channel number
1	-
20	1
2	2
21	3
3	4
22	5
4	6
23	7
5	8
24	9
6	10
25	11
7	12
26	13
8	14
27	15
9	16
28	17
10	18
29	19
11	20
30	21
12	22
31	23
13	24
32	Pat. GND
14	-
33	-
15	-
34	-
16	-
35	-
17	-
36	-
18	-
37	-
19	-

ANEXO C: NASA TASK LOAD INDEX

Name	Task	Date

Mental Demand — How mentally demanding was the task?

Very Low — Very High

Physical Demand — How physically demanding was the task?

Very Low — Very High

Temporal Demand — How hurried or rushed was the pace of the task?

Very Low — Very High

Performance — How successful were you in accomplishing what you were asked to do?

Perfect — Failure

Effort — How hard did you have to work to accomplish your level of performance?

Very Low — Very High

Frustration — How insecure, discouraged, irritated, stressed, and annoyed wereyou?

Very Low — Very High