Universidade de Brasília – UnB
Faculdade UnB Gama – FGA
Engenharia Eletrônica

Estudo da Implementação em Sistemas Embarcados de Algoritmos para Interfaces Cérebro-Máquina

Autor: Camila Carneiro Ferrer Santos
Orientador: Prof. Dr. Marcus Vinícius Chaffim Costa

Brasília, DF
2020

Camila Carneiro Ferrer Santos

Estudo da Implementação em Sistemas Embarcados de Algoritmos para Interfaces Cérebro-Máquina

Monografia submetida ao curso de graduação em Engenharia Eletrônica da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia Eletrônica.

Universidade de Brasília – UnB

Faculdade UnB Gama – FGA

Orientador: Prof. Dr. Marcus Vinícius Chaffim Costa

Brasília, DF

2020

Camila Carneiro Ferrer Santos
Estudo da Implementação em Sistemas Embarcados de Algoritmos para Interfaces Cérebro-Máquina/ Camila Carneiro Ferrer Santos. – Brasília, DF, 2020-
79 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Marcus Vinícius Chaffim Costa

Trabalho de Conclusão de Curso – Universidade de Brasília – UnB
Faculdade UnB Gama – FGA , 2020.

1. Interface Cérebro Máquina. 2. Sistemas embarcados. I. Prof. Dr. Marcus Vinícius Chaffim Costa. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Estudo da Implementação em Sistemas Embarcados de Algoritmos para Interfaces Cérebro-Máquina

CDU 02:141:005.6

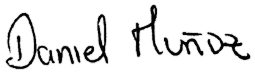
Camila Carneiro Ferrer Santos

Estudo da Implementação em Sistemas Embarcados de Algoritmos para Interfaces Cérebro-Máquina

Monografia submetida ao curso de graduação em Engenharia Eletrônica da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia Eletrônica.

Trabalho aprovado. Brasília, DF, 14 de dezembro de 2020 – Data da aprovação do trabalho:

Prof. Dr. Marcus Vinícius Chaffim Costa
Engenharia Eletrônica - Faculdade do Gama/Universidade de Brasília (FGA/UnB)
Orientador



Prof. Dr. Daniel Mauricio Muñoz Arboleda
Engenharia Eletrônica - Faculdade do Gama/Universidade de Brasília (FGA/UnB)
Convidado 1

Prof. Dr. Renato Coral Sampaio
Engenharia de Software - Faculdade do Gama/Universidade de Brasília (FGA/UnB)
Convidado 2

Brasília, DF
2020

Aos meus pais, Verônica e Valter, e minha irmã, Nicole,
meus melhores orientadores na vida que sempre apoiaram meus projetos pessoais.

Agradecimentos

A Deus, pela minha vida, e por me permitir ultrapassar todos os obstáculos encontrados ao longo da realização deste trabalho.

À minha família, que me incentivou nos momentos difíceis e compreendeu a minha ausência enquanto eu me dedicava à realização deste trabalho. À minha irmã em especial, por todo o apoio e pela ajuda, que muito contribuíram para a realização deste trabalho.

Aos professores, pelas correções e ensinamentos que me permitiram apresentar um melhor desempenho no meu processo de formação profissional ao longo do curso. Em especial, ao professor Marcus Chaffim, por ter sido meu orientador e ter desempenhado tal função com dedicação e amizade.

Resumo

Nas últimas décadas, as pesquisas relacionadas à Interface Cérebro-Máquina (BCI) estão em uma das áreas de maior crescimento da engenharia neural, pois possui potencial de comercialização nos mercados de saúde, pesquisa e consumo. A implementação de um sistema BCI necessita de um processamento complexo de sinais de eletroencefalografia (EEG) que inclui filtragem, extração de características e algoritmos de classificação e alguns desses classificadores não necessitam de uma execução com alta demanda computacional. Nesse sentido, o presente trabalho traz a proposta de utilizar a plataforma PocketBeagle para embarcar implementações de algoritmos para BCI utilizando a linguagem Python. Os algoritmos implementados utilizaram os classificadores de Análise Discriminante Linear (LDA) e Máquina de Vetores de Suporte (SVM) juntamente com filtro de Padrões Espaciais Comuns (CSP). Esse trabalho visou explorar algumas características do sistema embarcado para otimizar desempenho, consumo de recursos e tempo de treinamento dos algoritmos implementados. Reproduziu-se a implementação desenvolvida por Fabien Lotte e Cuntai Guan em um estudo publicado em 2011 com o *Data Set IVa* da *BCI Competition III* para identificar os tempos de treinamento para cada sujeito e confirmar a média de acurácia encontrada na pesquisa. Foram geradas matrizes de confusão com os resultados provenientes das execuções dos algoritmos em Python para cada sujeito e essas matrizes originaram as medidas de desempenho de acurácia, confiabilidade, sensibilidade e *f-score* para validação da plataforma PocketBeagle como um possível sistema para implementações de BCI. O algoritmo em linguagem Python com CSP e LDA implementado para este trabalho obteve um tempo de treinamento maior em 5,78 vezes ao compara com o tempo encontrado durante a reprodução da pesquisa de Lotte e Guan e o algoritmo em Python com CSP e SVM aumentou em 5,71 vezes. Ao embarcar as implementações em Python na PocketBeagle, esse tempo de treinamento aumentou para, em média, 42,98s para o LDA e 42,66s para o SVM. Ao comparar com a execução em computador pessoal, o tempo de treinamento do classificador LDA aumentou em 48,50 vezes e do classificador SVM em 48,14 vezes. Ao analisar os consumos de memória das implementações no sistema embarcado, observa-se que os códigos implementados consumiram menos da metade da memória disponível na PocketBeagle de 512MB, sendo o consumo do classificador LDA de 167MB em seu pico e do SVM de 177MB no pico da execução. Utilizando as métricas decorrentes da matriz de confusão, fica claro que o classificador SVM tem melhor desempenho do que o LDA, pois sua acurácia é de 83,14% e seu *f-score* é 0,8111, enquanto para o classificador LDA têm-se acurácia de 66,29% e *f-score* de 0,6940.

Palavras-chaves: Interface Cérebro-Máquina; Sistemas Embarcados; PocketBeagle; Python; Sistemas em Chip.

Abstract

Over the past decades, research involving Brain Computer Interface is one of the most growing field in neural engineering since it has the potential of commercialization in the health, research and consumption markets. The implementation of a BCI system requires a complex processing of EEG signals that includes filtering, extraction of characteristics and classification algorithms. Some of these classification algorithms do not require an execution with high computational performance. In this sense, the present work proposes to use the PocketBeagle embedded platform for implementation of these algorithms using the Python language. The implemented algorithms used the Linear Discriminant Analysis (LDA) and Support Vector Machine (SVM) classifiers together with the Common Spatial Patterns filter (CSP). This work aims to explore some characteristics of the embedded system to optimize performance, resource consumption and training time of the implemented algorithms. The implementation developed by Fabien Lotte and Cuntai Guan in a study published in 2011 was reproduced with the Data Set IVa of the BCI Competition III to identify the training times for each subject and confirm the average accuracy found in the study. Confusion matrices were generated with the results from the execution of the Python algorithms for each subject and these matrices originated the measures of accuracy, precision, recall and f-score to validate the PocketBeagle platform's performance as a possible system for BCI implementations. The Python language algorithm with CSP and LDA implemented for this work obtained a training time of 5.78 times greater when compared to the time found during the reproduction of the Lotte and Guan research and the Python algorithm with CSP and SVM increased by 5.71 times. When running the Python implementations on the PocketBeagle, this training time increased to an average of 42.98s for the LDA classifier and 42.66s for the SVM. When compared to running on notebook, the LDA classifier's training time increased by 48.50 times and the SVM classifier by 48.14 times. When analyzing the memory consumptions of the implementations in the embedded system, it is observed that the implemented codes consumed less than half of the available memory in the PocketBeagle with the consumption of the LDA classifier being 167MB at its peak and the SVM of 177MB. Using the metrics resulting from the confusion matrix, it is clear that the SVM classifier has better performance than the LDA, as its accuracy is 83.14% and its f-score is 0.8111, while the LDA classifier has an accuracy of 66.29% and f-score of 0.6940.

Key-words: Brain Computer Interface; Embedded Systems; PocketBeagle; Python; System on Chip.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

AR	*Autoregressive* (Auto-Regressivo)
BCI	*Brain–Computer Interface* (Interface Cérebro-Máquina)
BP	*Band Power* (Potência da Banda)
CSP	*Common Spatial Pattern* (Padrão Espacial Comum)
DDR	*Double Data Rate* (Taxa de Transferência Dobrada)
DFT	*Discrete Fourier Transform* (Transformada Discreta de Fourier)
DRE	Dessincronização Relacionada a Eventos
ECoG	Eletrocorticograma
EEG	Eletroencefalografia
FFT	*Fast Fourier Transform* (Transformada Rápida de Fourier)
fMRI	Ressonância Magnética Funcional
FN	*False negative* (Falso negativo)
FP	*False positive* (Falso positivo)
GPIO	*General Purpose Input Output* (Portas Programáveis de Entrada e Saída)
IIR	*Infinite impulse response* (Filtro de Resposta ao Impulso Infinito)
INR	*Intracortical Neuron Recording* (Gravação Intracortical de Neurônios)
LDA	*Linear Discriminant Analysis* (Análise Discriminante Linear)
LPFs	*Local Field Potentials* (Potenciais de Campo Local)
MEG	Magnetoencefalografia
MUA	*Multi-Unit Activity* (Atividade de Múltiplos Neurônios)
OTG	*USB On-the-Go*
PCB	*Printed Circuit Board* (Placa de Circuito Impresso)
PCLs	Potenciais Corticais Lentos

PEAEEs	Potencias Evocados Auditivos de Estado Estacionário
PESSEEs	Potenciais Evocados Somato-Sensitivos de Estado Estacionário
PET	Tomografia por Emissão de Pósitrons
PEVEEs	Potenciais Evocados Visuais de Estado Estacionário
RISC	*Reduced Instruction Set Computer* (Computador com Conjunto Reduzido de Instruções)
RSMs	Ritmos Sensorio-Motor
SBCs	*Single Board Computers* (Computadores de Placa Única)
SNC	Sistema Nervoso Central
SNP	Sistema Nervoso Periférico
SoC	*System on Chip* (Sistema em Chip)
SQUID	*Superconducting Quantum Interference Device* (Dispositivo Supercondutor de Interferência Quântica)
SRE	Sincronização Relacionada a Eventos
SVM	*Support Vector Machines* (Máquinas de Vetor de Suporte)
SUA	*Single Unit Activity* (Atividade de um Neurônio)
TN	*True negative* (Verdadeiro negativo)
TP	*True positive* (Verdadeiro positivo)
UART	*Universal Asynchronous Receiver/Transmitter* (Transmissor/Receptor Assíncrono Universal)
USB	*Universal Serial Bus* (Porta Serial Universal)

Lista de símbolos

Λ	matriz de covariância
λ	autovalor
μ	média das distribuições normais
Σ	estimativa da matriz de covariância
ξ	variável de folga

Sumário

1 Introdução

O cérebro humano é uma intricada rede de conexões repleta de mistérios devido à sua vasta complexidade. Este órgão é responsável pela nossa inteligência, pela interpretação das sensações, pelo movimento do corpo e pelo controle de todos os nossos comportamentos e, devido a isso, a grande maioria dos neurocientistas concentram suas pesquisas no cérebro (Psychology Career Center, 2014).

Ao longo de centenas de anos, cientistas aprenderam muito sobre o cérebro, incluindo os incontáveis métodos pelos quais a informação é transportada de um neurônio para outro em uma ação chamada sinapse e como centenas, ou milhares, de neurônios podem funcionar juntos para formar um circuito maior responsável por qualquer atividade. Todos os cérebros humanos compartilham circuitos anatômicos básicos e interações sinápticas, mas o padrão preciso de conexões e interações é altamente variável de pessoa para pessoa (BRAIN Initiative, 2016).

Os pesquisadores continuam a explorar a mecânica por trás de um cérebro saudável que funciona rápida e automaticamente na velocidade do pensamento. Para compreender verdadeiramente esse circuito, é necessário identificar e caracterizar as células componentes, definindo suas conexões umas com as outras, monitorando e registrando seus padrões de atividade. Ainda estamos no ponto em que ainda há muito trabalho para identificar as principais diferenças em uma vasta gama de neurônios desconhecidos (BRAIN Initiative, 2016).

Uma revolução neurocientífica foi impulsionada pelos avanços científicos e tecnológicos como a invenção do microscópio, que possibilitou o estudo detalhado da anatomia desse órgão, e a compreensão da eletricidade, que permitiu o reconhecimento da dinâmica e atividades do cérebro (CARTER, 2014).

No início dos estudos, a única maneira de relacionar funções como visão, emoção ou fala com os locais do cérebro onde eles ocorrem era encontrar uma pessoa que tivesse dificuldades de exercer tal ação devido a alguma lesão cerebral e então esperar até que eles falecessem para localizar e mensurar a extensão do dano cerebral (CARTER, 2014). Caso contrário, era possível apenas pressupor o que estava acontecendo com o cérebro observando o comportamento dessas pessoas.

Os primeiros mapas detalhados das funções do cérebro humano foram feitos pelo neurocirurgião canadense Wilder Penfield e seus colaboradores, Edwin Boldrey e Theodore Rasmussen (PENFIELD; BOLDREY, 1937). Penfield e seus colegas passaram a experimentar a estimulação elétrica de diferentes áreas do cérebro de pacientes conscientes submetidos a cirurgias de cérebro aberto para controlar a epilepsia e assim foram capazes de produzir os mapas topográficos do cérebro e seus homúnculos correspondentes.

Eles não foram os primeiros cientistas a tentar materializar a função do cérebro humano por meio de um homúnculo, porém, foram os primeiros a diferenciar entre função sensorial e motora e mapear os dois através do cérebro separadamente. Seus desenhos tornaram-se talvez os mapas conceituais mais famosos da neurociência moderna, porque ilustraram os dados de maneira convincente em um único olhar (PENFIELD; BOLDREY, 1937).

Hoje, modernas técnicas de imagem permitem aos neurocientistas enxergar a atividade elétrica no cérebro enquanto uma pessoa realiza várias tarefas ou processos de pensamentos (TOGA; MAZZIOTTA, 2000). Essas técnicas proporcionaram um rápido avanço no entendimento do cérebro e produzem um mapa cada vez mais detalhado de suas funções realizando análises estrutural, que verifica a anatomia do cérebro, e funcional, que tenta medir e localizar a atividade cerebral.

A imagem que surge desses estudos revela que o cérebro é um sistema surpreendentemente complexo e sensível, no qual cada parte afeta todas as outras, e a investigação neurocientífica do cérebro é um trabalho ainda em andamento (CARTER, 2014).

A capacidade integrativa dos mapas cerebrais permite a inclusão de um conjunto diversificado de observações e eles são usados para descrever a estrutura, a função e a conectividade do cérebro (TOGA; MAZZIOTTA, 2000). Com o mapeamento cerebral através do uso de sensores para monitoramento, catalogou-se a base de conhecimento dos sistemas nervosos e mostrou subsistemas e circuitos detalhados do cérebro.

Dentro da análise funcional, algumas das técnicas utilizam a tomografia por emissão de pósitrons (PET) e a ressonância magnética funcional (fMRI) examinando mudanças regionais no metabolismo cerebral e no fluxo sanguíneo para medir a atividade neuronal (CRONE et al., 1999). Técnicas como eletroencefalografia (EEG) e magnetoencefalografia (MEG) têm sido usadas no córtex diretamente para medição.

Os equipamentos de EEG estão se tornando mais disponíveis no mercado devido diminuição de seu custo e aumento de sua precisão ao medir a atividade das ondas cerebrais na camada externa do cérebro. Eletrodos sensíveis são conectados à cabeça e os sinais são amplificados para fornecer um gráfico de potencial elétrico que pode ser medido e comparado em diferentes pontos da cabeça para fornecer um mapa de atividade bidimensional do córtex cerebral (POWLEDGE, 1997).

Para mapear essas atividades, ondas cerebrais emitidas durante um estímulo específico são gravadas e o experimento é repetido diversas vezes. Os gráficos são então calculados e os dados resultantes são chamados de potenciais relacionados a eventos. Estes foram importantes nos anos 60 e 70, quando os pesquisadores estavam tentando descobrir se certas regiões do córtex cerebral eram especializadas para realizar algumas tarefas (MULDER, 2007).

Diversos estudos indicaram que a execução mental de um movimento sem qualquer ação física sendo efetivamente realizada, ou sem qualquer ativação periférica muscular, pode resultar nas mesmas áreas cerebrais sendo ativadas que as de uma execução física real (MULDER, 2007).

A atividade cerebral, associada à imagem motora, pode ser usada em interfaces cérebro-computador (BCI). BCI são tecnologias, invasivas ou não invasivas, que permitem que vários sinais neurofisiológicos sejam transformados em comandos para um equipamento externo ou computador (MOKIENKO et al., 2014).

Esta tecnologia tem estado em constante desenvolvimento nos últimos anos para uso na reabilitação de pessoas com doenças neurológicas ou com síndrome do encarceramento por fornecer um sistema de interações com o mundo à sua volta (SHIH; KRUSIESKI; WOLPAW, 2012).

Essas interfaces também permitem que sujeitos com distúrbios motores possam controlar próteses robóticas, cadeiras de rodas e além outros dispositivos que os auxiliem na execução de atividades.

Porém o desenvolvimento de BCI atualmente não está mais restrito apenas a pessoas com doenças ou em tratamentos, há uma mudança de foco para pessoas saudáveis também. Especialmente a indústria de entretenimento está se tornando um mercado favorável, pois os usuário provavelmente se adaptariam rapidamente ao uso do EEG como uma nova modalidade dando-lhes vantagens ou novas experiências (LARSEN, 2007).

Ao levar a BCI ao nível do entretenimento, a motivação para tornar os sistemas disponíveis mais acessíveis, mais rápidos, mais baratos e universais terá uma prioridade muito maior, já que os sistemas atuais não atendem a tais padrões.

Enquanto muitos estudos iniciais de BCI buscavam satisfazer requisitos de operação e *feedback* para o usuário em tempo real (WOLPAW et al., 2002), estudos mais recentes estão utilizando sofisticados algoritmos de reconhecimento e classificação de padrões que convertem a atividade neural em sinais de controle necessários (LOTTE et al., 2007).

A crescente popularidade de pesquisas relacionadas a BCI provém da facilidade de se realizar análises a partir de sinais já divulgados abertamente. Essas pesquisas tem se concentrado fortemente no desenvolvimento de técnicas de processamento de sinais e aprendizado de máquina para classificar com precisão a atividade neural (GRAIMANN; ALLISON; PFURTSCHELLER, 2010).

No entanto, a implementação de um sistema BCI necessita de um processamento complexo de sinais de EEG incluindo filtragem, extração de características e algoritmos de classificação.

Pela maioria dos sistemas de BCI serem desenvolvidos em computadores pessoais, existe um grande interesse do mercado em implementar a BCI em plataformas embarcadas que atendam às especificações do sistema em termos de tempo de resposta, custo-eficácia, consumo de energia e precisão (BELWAFI et al., 2018).

1.1 Justificativa

As pesquisas relacionadas à interface cérebro-máquina estão em uma das áreas de maior crescimento da engenharia neural, com potencial de comercialização nos mercados de saúde, pesquisa e consumo (DALY; HUGGINS, 2015).

No entanto, apesar do grande desenvolvimento recente de interfaces, muitos problemas ainda precisam ser resolvidos para o projeto do sistema, como algoritmos com maior precisão da classificação, redução de tempo do treinamento do usuário e otimização da robustez e a segurança.

Com a disseminação de poderosas ferramentas de processamento e sistemas eletrônicos potentes e acessíveis, o processamento do sinal biológico foi facilitado, abrindo caminho ao desenvolvimento de sistemas que permitem a restauração das capacidades motoras de pessoas com graves desordens ou deficiências, facilitando suas atividades diárias e dando melhor qualidade de vida.

Ademais, as interfaces convencionais costumam ser caras, complexas de operar e carecem de portabilidade, o que limita seu uso em laboratórios, dessa forma BCIs portáteis e de baixo custo podem amenizar esses problemas.

Portanto, desenvolver uma BCI portátil, de alto desempenho e de baixo custo é de grande importância para o avanço desse campo de pesquisa, sendo necessário comparar seu desempenho com o de outras BCIs convencionais para avaliar sua eficiência.

A motivação principal deste trabalho está em explorar os recursos da plataforma embarcada PocketBeagle na implementação de uma interface, visando a utilização de algumas das características mais importantes dessa plataforma como flexibilidade, facilidade de desenvolvimento, eficiência computacional, custo reduzido e baixo consumo energético.

1.2 Objetivos

1.2.1 Objetivo geral

O presente trabalho tem como objetivo geral desenvolver uma implementação em sistema embarcado de algoritmos de treinamento utilizando Padrões Espaciais Comuns (CSP) com Análise Discriminante Linear (LDA) e Máquina de Vetores de Suporte (SVM)

para interfaces cérebro-máquina, buscando estudar os ganhos do tempo de execução em comparação a implementações tradicionais.

1.2.2 Objetivos específicos

- Reproduzir os resultados obtidos por Lotte e Guan (2011) no desenvolvimento do algoritmo de treinamento;
- Implementar algoritmos utilizando a linguagem Python;
- Embarcar na plataforma *PocketBeagle* os algoritmos;
- Validar as implementações utilizando as bases de dados do *BCI Competition III dataset IVa*;
- Validar as implementações utilizando métricas de desempenho;
- Analisar o tempo de treinamento das implementações embarcadas;
- Analisar o consumo de memória das implementações embarcadas.

1.3 Estrutura do trabalho

Este trabalho visa a realização de atividades teóricas e experimentais, cuja finalidade principal é a aquisição de novos conhecimentos sobre sistemas de processamento embarcado de algoritmos de BCI.

O trabalho divide-se em cinco capítulos denominados Introdução (Capítulo 1), Referencial Teórico (Capítulo 2), Materiais e Métodos (Capítulo 3), Resultados e Discussão (Capítulo 4) e Conclusões (Capítulo 5), sendo estes divididos em três partes.

A primeira parte contém o Capítulo 1 e o Capítulo 2 que lidam com a abordagem geral do estudo sobre o tema em questão e apresentam os objetivos que se deseja alcançar ao longo do trabalho. Além de apresentar alguns conceitos sobre o processamento e análise do algoritmo que será aplicados nas metodologias propostas.

A segunda parte envolve o Capítulo 3 que apresenta os materiais, procedimentos e métodos a serem utilizados para desenvolvimento deste estudo, detalhando as metodologias utilizadas na implementação, suas ferramentas e características dos materiais.

Já na terceira parte, que contém os capítulos Capítulo 4 e Capítulo 5, apresenta uma análise comparativa dos resultados do trabalho obtidos pela reprodução do algoritmo de treinamento, implementação e otimização dos algoritmos com LDA e SVM e análise dos resultados obtidos, mostrando também, as conclusões que se pôde encontrar e sugestões de trabalhos futuros nesse tema.

2 Referencial Teórico

2.1 Sistema Nervoso Central

O sistema nervoso humano pode ser dividido em dois sistemas: o Sistema Nervoso Central (SNC) e o Sistema Nervoso Periférico (SNP). O SNP é composto pelo sistema nervoso somático, que envolve neurônios conectados aos músculos esqueléticos, pele e órgãos sensoriais, e pelo sistema nervoso autônomo, que envolve neurônios que controlam as funções viscerais (CLARK; BOUTROS; MENDEZ, 2005).

O SNC é composto pelo cérebro e pela medula espinhal. A medula espinhal é o principal caminho que transmite os sinais do cérebro de controle para os músculos em todo o corpo e recebe os sinais sensoriais dos músculos e da pele de volta para cérebro (CLARK; BOUTROS; MENDEZ, 2005).

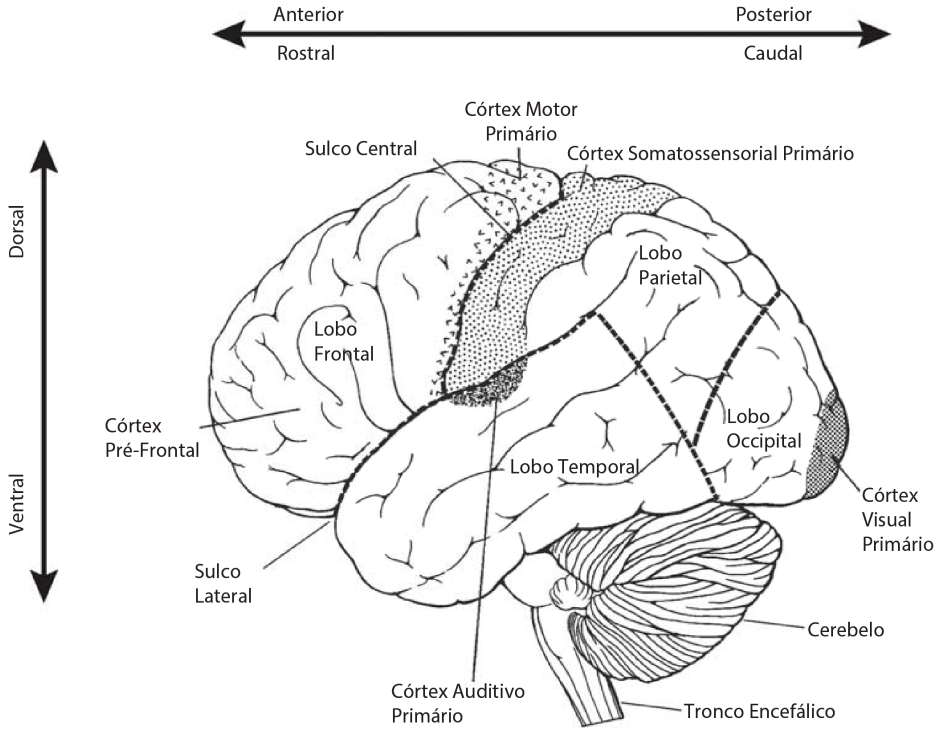


Figura 1 – Principais divisões do córtex cerebral humano. Adaptado de Wolpaw e Wolpaw (2012)

Na Figura 1 pode-se observar que o córtex cerebral humano é composto por regiões anatômicas distintas separadas por sulcos e giros. Abaixo do córtex, existem outras estruturas mais profundas denominadas áreas subcorticais, que incluem os gânglios da base, o cerebelo, o tronco encefálico e o tálamo (CARTER, 2014).

O córtex cerebral tem quatro partes principais, ou lobos, denominadas frontal, parietal, occipital e temporal, e é responsável pelo planejamento do movimento. Por ser relativamente acessível experimentalmente, é a área do cérebro que tem foco principal na maioria das pesquisas da BCI (WOLPAW; WOLPAW, 2012).

Na base do cérebro, o tronco encefálico, composto por mesencéfalo, ponte e bulbo, interpõe-se entre a medula espinal e o diencéfalo sendo o responsável por transmitir toda a informação do cérebro para o resto do corpo (CARTER, 2014).

Um neurônio é um tipo de célula que é geralmente considerada como a unidade computacional básica do sistema nervoso. Quando o neurônio recebe impulsos fortes de outros neurônios, uma cascata de eventos químicos é se inicia, causando uma queda no potencial da membrana (CLARK; BOUTROS; MENDEZ, 2005).

Essa rápida ascensão e queda do potencial da membrana é chamada de potencial de ação ou *spike*, e representa o modo dominante de comunicação entre um neurônio e outro. O *spike* é um evento com pouca ou nenhuma informação na forma do potencial de ação em si, por isso a informação que se analisa é a taxa de disparos de *spikes* por segundo (RAO, 2013). Os neurônios são, portanto, frequentemente modelados computacionalmente como emitindo uma saída digital 0 ou 1, como mostra a Figura 2.

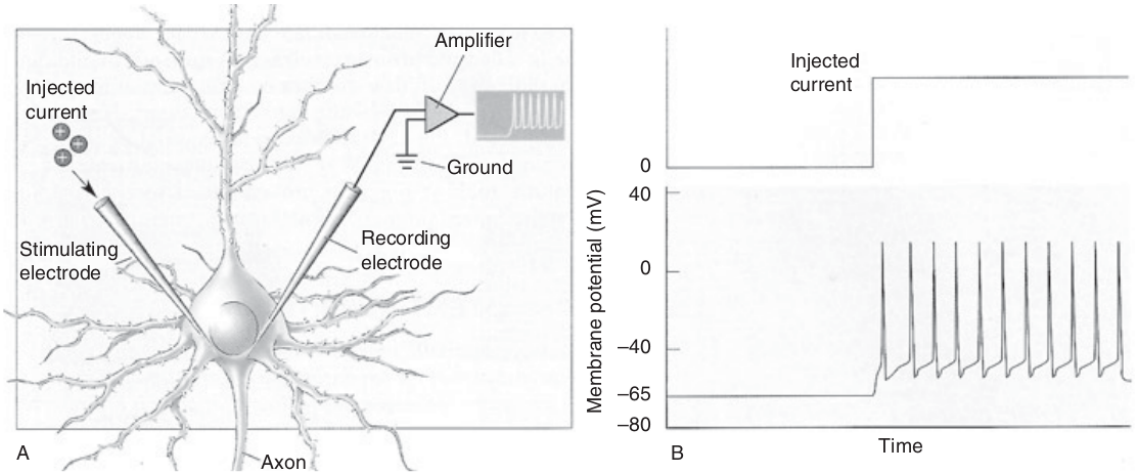


Figura 2 – Geração de *spikes* ao injetar íons positivos. Fonte: Nam, Nijholt e Lotte (2018)

Verifica-se ainda na Figura 2 um procedimento experimental que injeta uma corrente (íons positivos) no corpo celular de um neurônio usando um eletrodo estimulante e registra a mudança no potencial de membrana da célula usando um eletrodo de registro. No gráfico ao lado observa-se o resultado da injeção de corrente que resulta em uma sequência de potenciais de ação. Cada pico tem uma forma estereotipada que rapidamente aumenta e cai novamente (BEAR; CONNORS; PARADISO, 2007).

Na década de 1940 o neurocirurgião canadense Wilder Penfield (1891-1976) observou pacientes com epilepsia intratável clinicamente e pensou que se pudesse induzir

as crises através de uma leve eletroestimulação no córtex cerebral, encontraria o foco epileptogênico e a remoção dessa região levaria à cura (PENFIELD; BOLDREY, 1937).

Com os pacientes conscientes e utilizando apenas anestesia local, Penfield realizava a craniotomia, localizava os focos das crises e os removia, porém aproveitava para estimular outras regiões do córtex e observar os efeitos motores e sensitivos, assim como respostas cognitivas complexas envolvendo vários sentidos (PENFIELD; BOLDREY, 1937).

Com esses experimentos, ele identificou relações de algumas áreas do córtex cerebral com as diversas regiões do corpo humano e que havia uma proporcionalidade do tamanho destas áreas corticais com as funções periféricas (SILVA, 2013).

A partir desses dados, foi possível desenvolver um mapa cerebral, no qual diferentes regiões corporais eram representadas no córtex, com seus tamanhos e disposições configurados de acordo como eram encontradas no corpo durante o experimento, idealizando, assim, o Homúnculo de Penfield mostrado na Figura 3.

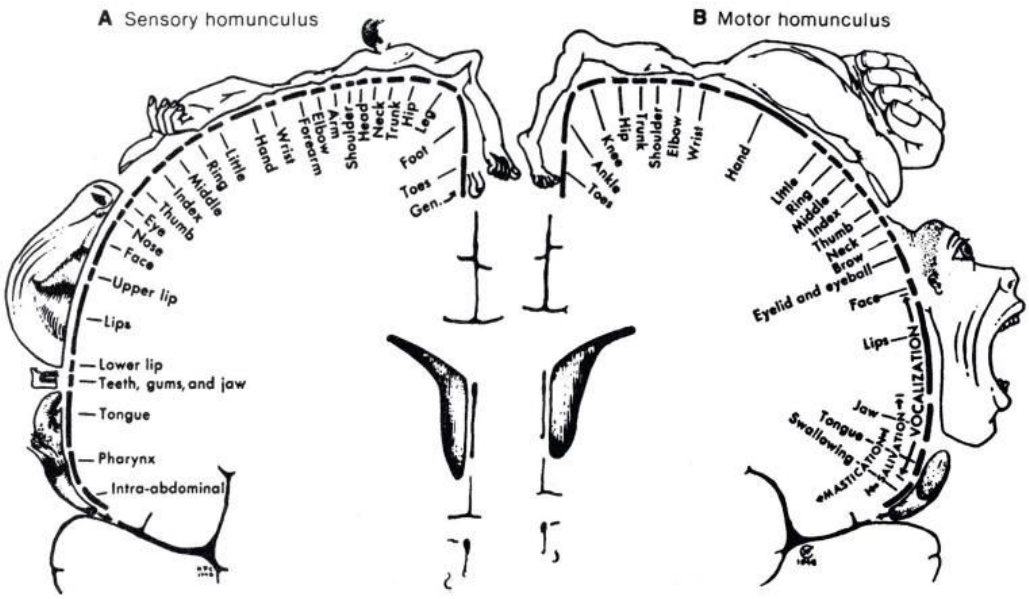


Figura 3 – Homúnculo de Penfield, sensorial e motor. Fonte: Montessori Public (2016)

2.2 Interface Cérebro-Máquina

A BCI é um sistema que mede a atividade do SNC e a converte em uma saída artificial que substitui, restaura, aprimora ou complementa a atividade do SNC e seu componente de interface neural é o dispositivo de hardware que detecta sinais cerebrais para que eles possam ser enviados para os outros componentes da interface para análise e conversão em comandos úteis. As formas de detecção do sinal neural podem ser categorizadas em dois grupos principais: Invasivo e não invasivo como mostra a Figura 4.

Os métodos invasivos necessitam de cirurgia para a implantação dos sensores que realizam a leitura da atividade elétrica do cérebro, entretanto nos métodos não invasivos não há necessidade de realizar uma cirurgia ou mesmo romper a pele, pois dependem de sensores colocados sobre a cabeça para medir essa atividade (LARSEN, 2007).

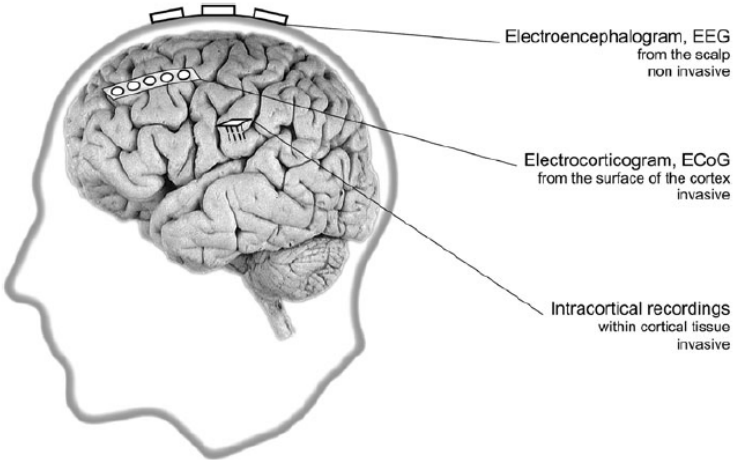


Figura 4 – Métodos de detecção da atividade elétrica do cérebro. Fonte: Graimann, Allison e Pfurtscheller (2010)

Técnicas invasivas combinam excelente qualidade de sinal, ótima resolução espacial e uma faixa de frequência mais alta, além da reaplicação de eletrodos ser desnecessária para essa abordagem (GRAIMANN; ALLISON; PFURTSCHELLER, 2010).

A gravação intracortical de neurônios (INR) é uma técnica que permite registrar a atividade neuronal na massa cinzenta do cérebro a partir dos impulsos elétricos do cérebro. Através do uso de um eletrodo penetrante feito de vidro, platina ou tungstênio colocado próximo ou dentro de um neurônio, as correntes elétricas podem ser observadas (NAM; NIJHOLT; LOTTE, 2018).

Essa técnica pode ser tão precisa que detecta a atividade de um neurônio (SUA) ou a atividade de múltiplos neurônios (MUA), porém, geralmente é utilizada para identificar potenciais de campo local (LPFs), que são os impulsos elétricos na área circundante da colocação de eletrodos, e a técnica é aplicada especialmente em sujeitos com distúrbios motores severos (PERGE et al., 2013).

A resolução espacial dessa técnica é muito detalhada e supera todos os outros tipos de técnicas invasivas e não invasivas de neuroimagem, porém possui riscos associados, incluindo a diminuição da aquisição de sinal ao longo do tempo, dano tecidual, rejeição de corpos estranhos ou movimento de eletrodos no cérebro (GUNASEKERA et al., 2015).

O eletrocorticograma (ECoG), também chamado de EEG intra cranial, é um método de registro de impulsos elétricos com eletrodos que são colocados no cérebro para contornar materiais que possam impedir a leitura dos sinais, como o couro cabeludo e o crânio. O ECoG oferece maior resolução temporal e espacial do que o EEG com largura

de banda mais ampla, maior amplitude característica e muito menos vulnerabilidade a artefatos como eletromiografia ou ruído (NAM; NIJHOLT; LOTTE, 2018).

A fisiologia por trás do ECoG é a mesma que para o EEG, mas a sensibilidade no ECoG é maior devido à proximidade dos eletrodos aos neurônios. Os eletrodos que são colocados levemente na camada epidural ou subdural do cérebro espaçados de 1 cm com espaçamento e agrupamento mantidos consistentes através do uso de estrutura de grade flexível (NAM; NIJHOLT; LOTTE, 2018).

Os métodos invasivos têm algumas vantagens claras, porém possuem uma inconveniência ao exigir neurocirurgia, tornando difícil a pesquisa sobre BCIs invasivas devido ao custo e ao risco de infecção pós-operatória e movimentação dos eletrodos de sua colocação inicial. Além disso, ainda não é claro se esses métodos podem fornecer gravações seguras e estáveis ao longo dos anos, pois podem acontecer reações teciduais que levem à deterioração da qualidade do sinal ou até mesmo à completa falha do eletrodo (GRAIMANN; ALLISON; PFURTSCHELLER, 2010).

Em 1924, Hans Berger descobriu que os sinais elétricos produzidos pelo cérebro humano poderiam ser registrados no couro cabeludo. Depois de cinco anos de estudo, publicou uma série de artigos que estabeleceram a eletroencefalografia como ferramenta principal para diagnóstico clínico e para pesquisa cerebral (WOLPAW; WOLPAW, 2012).

Após criar a primeira máquina de gravação de EEG, Berger sugeriu que as correntes cerebrais mudavam dependendo do status funcional do cérebro como sono, anestesia e epilepsia. Essa sugestão acabou criando um novo ramo da ciência médica chamado neurofisiologia (HAZARIKA et al., 1997).

Durante o EEG, vários eletrodos são colocados na superfície do couro cabeludo com colas temporárias e cada eletrodo é conectado a um amplificador (SIULY; LI; ZHANG, 2016). Finalmente, os sinais elétricos do cérebro são convertidos em linhas onduladas na tela do computador para registrar os resultados, como mostra a Figura 5.

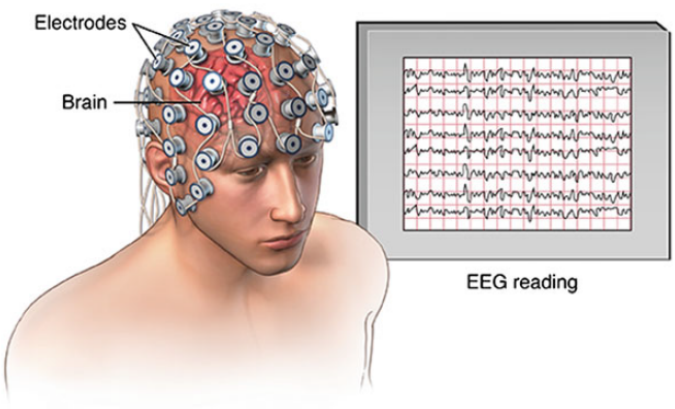


Figura 5 – Gravação de sinais de EEG. Fonte: Siuly, Li e Zhang (2016)

Dependendo do seu uso, as gravações de EEG podem utilizar de 1 a 256 eletrodos em paralelo, onde um par de eletrodos geralmente forma um canal que produz um sinal durante uma gravação EEG. A amplitude de um sinal EEG normalmente varia de cerca de 1 a 100 μV em um adulto normal, pois como a arquitetura do cérebro não é uniforme e o córtex é funcionalmente organizado, o EEG pode variar dependendo da localização dos eletrodos de registro (SIULY; LI; ZHANG, 2016).

A questão de como colocar os eletrodos é importante, porque diferentes lóbulos do córtex cerebral são responsáveis pelo processamento de diferentes tipos de atividades. Para obter gravações consistentes de regiões específicas da cabeça, é amplamente utilizado um sistema padrão para a colocação precisa de eletrodos, o que é chamado de Sistema Internacional 10-20 (GRAIMANN; ALLISON; PFURTSCHELLER, 2010).

O nome 10-20 indica as distâncias reais entre os eletrodos vizinhos que são 10% ou 20% da distância total do crânio nas direções anteroposterior (da frente para trás) e transversal (da esquerda para a direita). As posições são determinadas pelos pontos *nasion*, que é o ponto entre a testa e o nariz ao nível dos olhos, e *inon*, que é a proeminência óssea no crânio na parte de trás da cabeça (SIULY; LI; ZHANG, 2016).

A Figura 6 mostra que cada local usa uma letra para identificar o lobo e um número para identificar a localização no hemisfério. As letras F, T, C, P e O representam os lobos Frontal, Temporal, Central, Parietal e Occipital, respectivamente. O "Z"refere-se a linha média, números pares às posições dos eletrodos no hemisfério direito e números ímpares àqueles no hemisfério esquerdo (NAM; NIJHOLT; LOTTE, 2018).

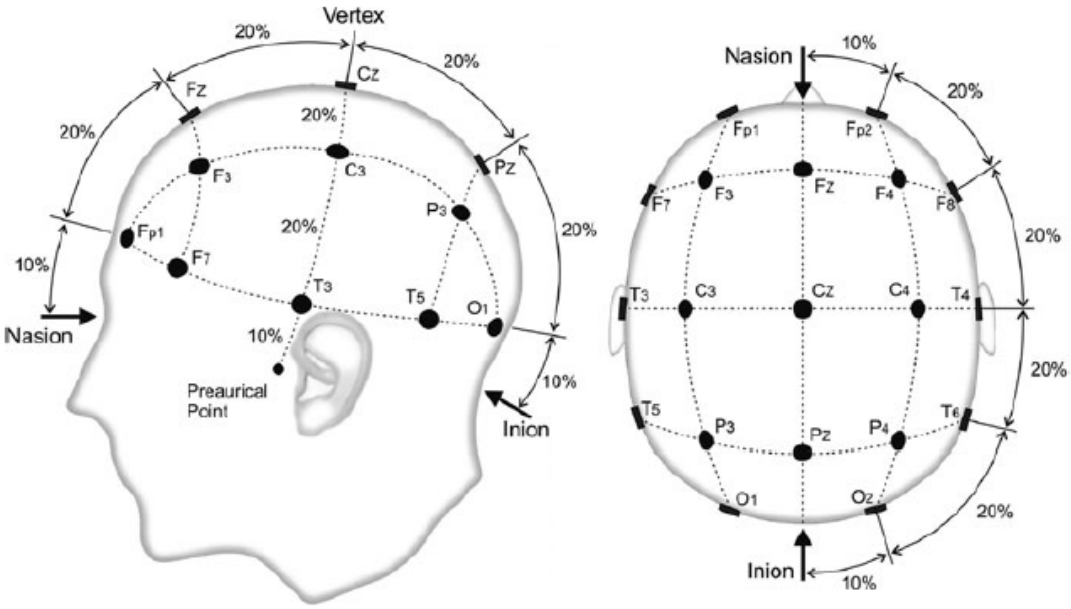


Figura 6 – Representação do Sistema Internacional 10-20. Fonte: Graimann, Allison e Pfurtscheller (2010)

O equipamento de EEG é barato, relativamente fácil de aplicar e possui uma ótima capacidade de detectar alterações dentro de um determinado intervalo de tempo. Alterações na atividade elétrica do cérebro ocorrem muito rapidamente e uma resolução extremamente alta é necessária para determinar precisamente os momentos em que esses eventos elétricos ocorrem (GRAIMANN; ALLISON; PFURTSCHELLER, 2010).

Além disso, os eletrodos de EEG são presos no couro cabeludo não necessitando de um procedimento invasivo para permitir que o pesquisador tenha acesso a um cérebro humano saudável (KAVITHA; KRISHNAVENI, 2015).

A principal desvantagem das gravações de EEG é a baixa resolução espacial devido às gravações serem realizadas no couro cabeludo, pois o sinal recebido é a soma do campo elétrico que é produzido por uma grande população de neurônios (KAVITHA; KRISHNAVENI, 2015). Como resultado, a atividade elétrica pode ser captada por vários eletrodos vizinhos.

Os métodos não invasivos podem ser usados para determinar o nível geral de atividade de uma determinada parte do cérebro sem precisar medir a atividade de cada neurônio individualmente. Devido aos inúmeros sistemas BCI registrarem distintos sinais cerebrais de diferentes áreas em diversas resoluções de forma não invasiva ou invasiva, é inevitável que os projetos e requisitos funcionais de uma BCI variem de acordo com o uso pretendido da interface e com o público-alvo (WOLPAW; WOLPAW, 2012).

A estratégia típica na engenharia de uma interface neural é buscar o tipo menos invasivo que seja capaz de fornecer sinais cerebrais com conteúdo de informação suficientes para ter segurança e eficácia de custos necessários (HAMALEINEN et al., 1993).

A maioria das BCIs registram os sinais eletromagnéticos de forma não invasiva a partir de sensores localizados acima do couro cabeludo. Os dois principais métodos extracranianos não invasivos usados para as BCIs são a eletroencefalografia e a MEG (WOLPAW; WOLPAW, 2012).

Além da geração de sinais elétricos, as fontes de corrente cerebral também podem gerar um campo magnético externo que pode ser detectado com sensores especializados. A detecção desses sinais magnéticos é realizada pelo MEG que registra o pequeno campo magnético gerado pelo cérebro usando um magnetômetro supercondutor de dispositivo de interferência quântica (SQUID) (HAMALEINEN et al., 1993).

Quando comparado ao EEG, o MEG tem vantagens e desvantagens. Para aplicações BCI, uma grande desvantagem do MEG é que os campos magnéticos associados às fontes de corrente cerebral são muito pequenos em relação às variações do campo magnético do ambiente que estão fora do controle experimental, tendo os problemas usuais de ruído dos campos da linha de energia (GRAIMANN; ALLISON; PFURTSCHELLER, 2010).

2.3 Processamento de sinais de EEG

A fim de melhorar o desempenho do projeto do sistema BCI, é necessário usar um bom método de processamento de sinal para permitir a extração mais fácil de características fisiológicas e além de um bom classificador adaptado às especificidades do sistema BCI (ZHANG et al., 2018).

Para obter uma alta precisão de classificação, os sinais EEG brutos devem ser pré-processados antes da extração do recurso devido à baixa relação sinal-ruído. Três etapas são necessárias para atingir esse objetivo: referência, filtragem temporal e aprimoramento de sinal (SOMERSET et al., 2010).

Como a tensão da atividade cerebral medida por um dado eletrodo é uma medida relativa, ela pode ser comparada a tensão cerebral de referência situada em outro local, resultando em uma combinação de atividade cerebral no eletrodo, atividade cerebral no local de referência e ruído. Devido a esse fator, o local de referência deve ser escolhido de tal forma que a atividade cerebral nesse local seja quase zero (SHARMILA, 2015).

Os sinais cerebrais estão naturalmente contaminados por muitos ruídos, porém eles podem ser removidos usando filtros simples devido à informação relevante ser encontrada em frequências abaixo de 30Hz (SIULY; LI; ZHANG, 2016). Portanto, todos os ruídos com frequências mais altas podem ser removidos usando um filtro passa-baixa.

Vários métodos de extração de características de sinal e algoritmos de classificação para o reconhecimento de padrões de EEG foram propostos ao longo do tempo para melhorar a taxa de bits de informação e a precisão da classificação (ZHANG et al., 2018). Essas características dos sinais do EEG podem ser extraídas no espectro temporal, espacial ou de frequência.

Grande parte da atividade cerebral se manifesta como oscilações contínuas de amplitude e modulação de frequência e é por essa razão que características no espectro da frequência têm sido amplamente utilizadas no processamento de sinais devido à sua facilidade de aplicação, velocidade computacional e interpretação direta dos resultados (WOLPAW; WOLPAW, 2012).

Embora a transformada de Fourier seja o método mais comum de conversão do domínio do tempo para o domínio da frequência, existem diversas alternativas que possuem características particularmente desejáveis, dadas restrições específicas ou objetivos específicos, como *Band Power* (BP), Transformada Rápida de Fourier (FFT) e Modelo Auto-Regressivo (AR) (SOMERSET et al., 2010).

O BP é um dos métodos mais diretos e intuitivos para rastrear modulações de amplitude em uma determinada frequência. Isola-se a frequência de interesse filtrando o sinal com um filtro passa banda, assim é produzido um sinal que é em grande parte

senoidal. Em seguida, para produzir valores puramente positivos, o sinal é retificado pela saturação do sinal ou pelo cálculo do seu valor absoluto. Finalmente, os picos adjacentes são suavizados juntos via integração ou filtragem passa-baixa (SZACHEWICZ, 2013).

A FFT é uma implementação eficiente da Transformada Discreta de Fourier (DFT), que representa o espectro da frequência de um sinal digital com uma resolução de frequência de taxa de amostragem por pontos da FFT, onde o ponto FFT é um escalar selecionável que deve ser maior ou igual ao comprimento do sinal digital e é tipicamente escolhido como um valor de base 2 para eficiência computacional (ZHANG et al., 2018).

A modelagem AR é uma alternativa aos métodos baseados em Fourier para o cálculo do espectro de frequência de um sinal, pois modela a filtragem do ruído branco por meio de um filtro de resposta ao impulso infinito (IIR). Os pesos específicos do filtro IIR moldam a entrada de ruído branco para corresponder às características do sinal que está sendo modelado (CLERC; BOUGRAIN; LOTTE, 2016).

Tem sido proposto que filtrar um ruído branco com um filtro AR é adequado para a geração de EEG, já que o EEG é essencialmente uma mistura de fontes espaciais de disparo espontâneo e medidas em diferentes posições (WOLPAW; WOLPAW, 2012).

Como os pesos de filtro IIR definem o espectro do sinal, a modelagem AR tem potencial para alcançar uma resolução espectral mais alta para blocos de sinal mais curto do que a FFT, além da estrutura do filtro IIR modela com precisão os espectros com picos nítidos e distintos, que são comuns em sinais como o EEG (SIULY; LI; ZHANG, 2016).

Sinais como o EEG são normalmente não estacionários, o que impede o uso da análise de Fourier para esses casos. Uma maneira de resolver esse problema é realizar a análise de Fourier em janelas de período reduzido com uma técnica chamada Transformada de Fourier de Tempo Curto (STFT) (SIULY; LI; ZHANG, 2016).

No entanto, a STFT deixa em aberto a questão do tamanho da janela, pois pequenas janelas fornecem boa resolução temporal, mas baixa resolução espectral. Em contraste, grandes janelas fornecem uma melhor resolução espectral, porém baixa resolução temporal (RAO, 2013).

Essa percepção levou ao desenvolvimento de uma técnica chamada *Wavelet* que busca alcançar o melhor equilíbrio entre resolução temporal e resolução espectral ao utilizar funções básicas em escalas diferentes, permitindo que um sinal seja analisado em várias resoluções (WOLPAW; WOLPAW, 2012).

Desta forma, a *Wavelet* resolve uma grande desvantagem das técnicas convencionais de análise espectral por serem altamente dependentes do comprimento do segmento selecionado, da ordem do modelo e de outros parâmetros (RAO, 2013).

Para um determinado bloco de amostra, os métodos FFT e AR produzem ape-

nas uma representação dessas flutuações na respectiva frequência. Ao observar essa caixa isoladamente, não é possível determinar quando ocorre um pulso naquela frequência específica dentro do bloco de amostragem. A *Wavelet* resolve esse problema produzindo uma representação de tempo-frequência do sinal (ZHANG et al., 2018).

A saída da FFT pode ser gerada a partir de um banco de filtros de banda passante em paralelos, com cada filtro centrado em intervalos de frequência uniformes. Em contrapartida, as *Wavelet* projetam um banco de filtros para obter uma resolução aprimorada de tempo-frequência (CLERC; BOUGRAIN; LOTTE, 2016).

Esta modelagem resulta em uma melhor resolução comparada à FFT porque mudanças em características de alta frequência podem ser identificadas em intervalos de tempo mais curtos que com o comprimento de segmento usado pela FFT (WOLPAW; WOLPAW, 2012).

2.4 Padrões de Sinais Cerebrais para Operação da BCI

Cada BCI é criada para responder a um certo tipo de sinal cerebral que mudam de acordo com sua base fisiológica. Devido a facilidade de aquisição do EEG comparado a outros sinais e sua abrangência de padrões cerebrais, esse sinal neuroelétrico é o mais utilizado em estudos de BCI (NIEDERMEYER; DA SILVA, 2005).

O potencial evocado P300 é um componente dos potenciais relacionados a eventos do EEG que atinge um pico positivo máximo de tensão em cerca de 300ms após o início de um estímulo. A amplitude da onda P300 é máxima nas regiões central e parietal do couro cabeludo e sua latência é proporcional à dificuldade de discriminar o estímulo-alvo do estímulo padrão, que podem ser visuais, auditivos ou táteis (RAO, 2013).

Como a resposta do P300 a estímulos externos é automática, o treinamento inicial não é necessário para ensinar os usuários a controlar seus sinais cerebrais. Os altos níveis de precisão combinados com o baixo custo e facilidade de uso dos EEGs para medir essa resposta tornam a onda P300 uma ferramenta útil e popular para as BCIs, proporcionando ao usuário uma grande variedade de modelos de interface (WOLPAW; WOLPAW, 2012).

Quando aplicados estímulos de natureza vibratória, ou seja, de estado estacionário, na área cortical associada irá aparecer atividade cerebral rítmica similar à frequência dos estímulos empregados que podem ser visuais, auditivos ou até mesmo táteis (RAO, 2013).

Os Potenciais Evocados Visuais de Estado Estacionário (PEVEEs) são atualmente a opção mais popular para sinais cerebrais nas operações da BCI, porém outros potenciais em estado estacionário também foram utilizados na pesquisa da BCI como os Potenciais Evocados Somato-Sensitivos (PESSEEs), que são gerados através de estímulos vibrotáteis, e os Potencias Evocados Auditivos (PEAEEs), gerados por estímulos auditivos também foram utilizados na pesquisa da BCI (NAM; NIJHOLT; LOTTE, 2018).

As BCIs que usam PEVEEs como seu sinal de controle geralmente têm luz ou outros estímulos que piscam em frequências diferentes ligados a uma opção de controle. As seleções das opções são feitas por meio do usuário que se concentra em qualquer estímulo associado à ação que deseja realizar, o dispositivo de neuroimagem registra a frequência dos sinais cerebrais e interpreta a seleção.

Esse tipo de BCI não requer treinamento e fornece uma comunicação mais rápida e confiável em EEG. No entanto, como as BCIs PEVEEs exigem olhar e foco nos olhos, elas podem não ser adequados para usuários com deficiências motoras graves e para pessoas com deficiência visual (NICOLAS-ALONSO; GOMEZ-GIL, 2012). Além disso, olhar por longos períodos de tempo para luzes piscando também pode induzir fadiga.

Os Ritmos Sensório-Motor (RSMs) são padrões de ondas cerebrais registrados nos córtices somatossensorial e motor e podem mudar devido a movimentos ou movimentos imaginários, sendo dois ritmos relevantes para RSMs: a banda Mu, que vai de 7Hz a 13Hz, e a banda Beta, de 14Hz a 30Hz (RAO, 2013).

O movimento real e imaginado cria o que é conhecido como Dessincronização Relacionada a Eventos (DRE), que é a diminuição da amplitude da banda de frequência nas áreas sensoriomotoras do cérebro relacionadas a movimentos ou movimentos imaginários, e Sincronização Relacionada a Eventos (SRE), que é o aumento na amplitude da banda de frequência nas áreas sensório-motoras imediatamente após o movimento ou movimento imaginado (GRAIMANN; ALLISON; PFURTSCHELLER, 2010).

O DRE da banda Mu começa logo antes do início do movimento, atinge o DRE máximo logo após o início do movimento e recupera seu nível original em poucos segundos. Em contraste, a banda beta mostra um DRE curto durante o início do movimento, seguido pelo SRE que atinge o máximo após a execução do movimento e este SRE ocorre quando o ritmo Mu ainda é atenuado (NICOLAS-ALONSO; GOMEZ-GIL, 2012).

Para que o sinal emitido por esses movimentos ou movimentos imaginários seja forte o suficiente, a área de uso no cérebro precisa ser grande o suficiente. Por isso, BCIs usando RSMs frequentemente usam o movimento imaginado de pés, mãos ou língua para fins de controle por serem representados em grandes áreas dos córtices somatossensoriais e motores devido ao movimento complexo e regular que produzem (GRAIMANN; ALLISON; PFURTSCHELLER, 2010).

Para esse tipo de padrão cerebral é necessário realizar treinamento extensivo empregando técnicas como o condicionamento operante, pois as ondas cerebrais e as interações com a BCI são controladas por processos de pensamento, além da estimulação externa não ser necessária para esse tipo de BCI (NAM; NIJHOLT; LOTTE, 2018).

Potenciais Corticais Lentos (PCLs) potenciais relacionados a eventos sensório-motores específicos que ocorrem em momentos previsíveis antes, durante ou após esses

eventos gerando mudanças nos níveis de polarização de neurônios corticais superficiais (WOLPAW; WOLPAW, 2012). Uma mudança na direção da polaridade negativa está associada ao aumento da atividade cortical e uma mudança na direção da polaridade positiva está associada à diminuição da atividade cortical.

Para utilizar PCLs é necessário treinamento extensivo e intensivo, usando estratégias cognitivas e comportamentais individualizadas e como os PCLs demoram de 1 segundo a vários segundos para se desenvolverem, a taxa de transferência de informações é bastante lenta em comparação com o PEVEEs e o P300 visual, o que não permite muita eficiência no uso. As BCIs que utilizam esse tipo de potencial não dependem de estímulos externos a fim de obter padrões de ondas cerebrais para influenciar a interface, em vez disso, os usuários controlam seus processos de pensamento para interagir com a BCI (NAM; NIJHOLT; LOTTE, 2018).

2.5 Classificação do sinal de EEG

2.5.1 Extração de características pelo filtro CSP

Devido à atividade simultânea de muitas fontes no cérebro, além de ruídos, a detecção de componentes relevantes da atividade cerebral é um desafio no processamento de sinais. Como os canais de EEG são altamente correlacionados, filtros espaciais são necessários para extrair informações localizadas com uma boa relação sinal/ruído (GRAIMANN; ALLISON; PFURTSCHELLER, 2010).

A utilização do CSP permite a identificação de filtros espaciais que maximizam a variância dos sinais de uma condição ao mesmo tempo que minimizam a variância dos sinais de outra condição. Uma vez que a variância dos sinais filtrados de banda passante é igual à potência de banda, os filtros CSP são adequados para detectar modulações de amplitude de ritmos sensório-motores e, consequentemente, discriminar estados mentais caracterizados por efeitos DRE e SRE (RAO, 2013).

Segundo Blankertz et al. (2008), dada uma entrada de dados $X \in \mathbb{R}^{C \times T}$, onde N é o número de canais e T o número de amostras no tempo por canal, o objetivo do CSP é encontrar M filtros espaciais, dados por uma matriz $W^{N \times M}$, onde cada coluna é um filtro espacial, que transforma linearmente os sinais de entrada de acordo com a Equação 2.1:

$$\mathbf{x}_{CSP}(t) = W^T \mathbf{x}(t) \tag{2.1}$$

onde $\mathbf{x}(t)$ é o vetor de sinais de entrada no tempo t de todos os canais. Para encontrar os filtros, as duas matrizes de covariância condicional a cada classe são estimadas

primeiro como mostra a Equação 2.2 (Blankertz et al., 2008).

$$R_c = \frac{1}{K} \sum_i X(X)^T \tag{2.2}$$

onde K para $c \in \{1, 2\}$ denota o tamanho do conjunto de índices correspondentes às tentativas pertencentes a cada classe. A técnica CSP envolve determinar uma matriz W tal como mostra a Equação 2.3 (Blankertz et al., 2008).

$$\begin{aligned} W^T R_1 W &= \Lambda_1 \\ W^T R_2 W &= \Lambda_2 \end{aligned} \tag{2.3}$$

onde Λ_c são matrizes diagonais para $c \in \{1, 2\}$ e $\Lambda_1 + \Lambda_2 = I$, onde I é a matriz identidade. Isso pode ser feito resolvendo um problema de autovalor generalizado dado pela Equação 2.4 (Blankertz et al., 2008).

$$R_1 \mathbf{w} = \lambda R_2 \mathbf{w} \tag{2.4}$$

Os autovetores generalizados $\mathbf{w} = \mathbf{w}_j$ que satisfazem a equação acima formam as colunas de W e representam os filtros espaciais CSP. Os autovalores generalizados $\lambda_1^j = \mathbf{w}_j^T R_1 \mathbf{w}_j$ e $\lambda_2^j = \mathbf{w}_j^T R_2 \mathbf{w}_j$ formam os elementos diagonais de λ_1 e λ_2 respectivamente (Blankertz et al., 2008).

Como $\lambda_1^j + \lambda_2^j = 1$, um valor alto para λ_1^j significa que a saída do filtro baseada no filtro $\mathbf{w}_j$ produz uma alta variação para sinais de entrada na classe 1 e uma baixa variação para sinais na classe 2 e vice-versa (Blankertz et al., 2008).

2.5.2 Métodos de Classificação

Um elemento importante nas operações da BCI é um classificador de dados, ou um algoritmo de classificação, que visa determinar automaticamente a intenção do usuário classificando os recursos extraídos do cérebro (RAO, 2013).

Existem diversas técnicas de classificação usadas para BCIs, porém três tipos de classificadores são comumente empregados para projetar sistemas BCI baseados em EEG: classificadores lineares, classificadores de redes neurais artificiais e classificadores de modelo oculto de Markov (CLERC; BOUGRAIN; LOTTE, 2016).

Os classificadores lineares são algoritmos discriminantes que utilizam uma função linear para classificar os dados e possuem simplicidade estrutural, precisão competitiva e treinamento e testes muito rápidos. O LDA e o SVM são as técnicas mais populares usadas para separar os dados que representam diferentes classes usando hiperplanos (SIULY; LI; ZHANG, 2016).

2.5.2.1 Classificador LDA

A técnica LDA possui várias vantagens que a tornam adequada para determinar a intenção do usuário da BCI como baixa exigência computacional e simplicidade de utilização e, devido a isso, gerou bons resultados para BCIs baseados em imagens motoras, soletrador P300 e BCIs assíncronas. No entanto, algumas limitações do LDA também devem ser observadas ao considerar essa abordagem para BCIs, pois suposições paramétricas e de linearidade restringem a análise e podem fornecer resultados pobres em dados complexos de EEG não-lineares (NAM; NIJHOLT; LOTTE, 2018).

O LDA projeta um vetor de entrada p-dimensional $\mathbf{x}$ em um hiperplano que divide o espaço de entrada em dois meios-espaços que representam uma classe ($+1$ ou -1). O limite de decisão é dado pela Equação 2.5 do hiperplano (RAO, 2013).

$$g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0 = 0 \tag{2.5}$$

O limite entre as duas classes é, portanto, caracterizado pelo vetor normal do hiperplano $\mathbf{w}$ e o limite w_0, que são determinados a partir dos dados de treinamento (RAO, 2013). Dado um novo vetor de entrada $\mathbf{x} \in X^p$, a classificação é obtida computando a Equação 2.6.

$$y = sign(\mathbf{w}^T\mathbf{x} + w_0) \tag{2.6}$$

que atribui $y = -1$ se $\mathbf{w}^T\mathbf{x} + w_0$ for negativo e $y = +1$ se $\mathbf{w}^T\mathbf{x} + w_0$ for positivo. Para calcular $\mathbf{w}$, o LDA assume que as distribuições condicionais de classe $P(x|c = 1)$ e $P(x|c = 2)$ são distribuições normais com média μ_c e covariância Λ_c para $c \in \{1, 2\}$ (RAO, 2013).

Uma classificação ótima acontece se ao atribuir entradas à primeira classe o logaritmo da razão de verossimilhança $log\left[\frac{P(x|c=1)}{P(x|c=2)}\right]$ está acima de um limite e, para a segunda classe, abaixo do limite. Como as duas distribuições são gaussianas, isso se reduz à comparação da Equação 2.7 (RAO, 2013).

$$(\mathbf{x} - \mu_1)^T\Sigma_1^{-1}(\mathbf{x} - \mu_1) - (\mathbf{x} - \mu_2)^T\Sigma_2^{-1}(\mathbf{x} - \mu_2) > K \tag{2.7}$$

onde K é o limite. Fazendo a suposição de que as covariâncias de classe são iguais, $\Sigma_1^{-1} = \Sigma_2^{-1} = \Sigma$, e têm classificação completa, obtêm-se o critério de classificação da Equação 2.8 (RAO, 2013).

$$\mathbf{w}^T\mathbf{x} > k \text{ onde } \mathbf{w} = \Sigma^{-1}(\mu_1 - \mu_2) \tag{2.8}$$

O limite k é frequentemente definido como estando no meio da projeção das duas médias de classe, isso é, $k = \frac{w^T(\mu_1-\mu_2)}{2}$. Pode ser mostrado que a escolha acima para w define um limite de decisão que maximiza a distância entre as médias m_1 e m_2 dos dados projetados de cada classe enquanto minimiza a variância dentro da classe dos dados projetados (RAO, 2013).

2.5.2.2 Classificador SVM

As SVMs lineares são amplamente utilizadas como classificadores para BCIs, pois produzem resultados de estado da arte na detecção de potenciais relacionados a eventos e imagens motoras. Suas vantagens são devido a razões teóricas como boas propriedades de generalização e insensibilidade relativa ao excesso de treinamento, porém suas desvantagens incluem um desempenho ruim se o número de recursos for muito maior que o número de amostras e uma baixa velocidade de execução (SIULY; LI; ZHANG, 2016).

Além de realizar classificações lineares, as SVMs podem ser estendidas para treinar classificadores não lineares usando o truque do kernel, que substitui o produto escalar por uma função de similaridade e, portanto, gera um limite de separação mais complexo. No entanto, apesar de alguns resultados muito encorajadores em competições, o uso de classificadores não lineares é frequentemente visto como desnecessário para as BCIs (CLERC; BOUGRAIN; LOTTE, 2016).

A SVM é uma ferramenta discriminante linear que maximiza a margem de separação entre duas classes com base na suposição de que melhora a capacidade de generalização do classificador. Em contraste, o LDA maximiza a margem média, ou seja, a margem entre as médias da classe (GRAIMANN; ALLISON; PFURTSCHELLER, 2010). A Figura 7 ilustra um hiperplano linear típico aprendido por um SVM.

Segundo Graimann, Allison e Pfurtscheller (2010), um classificador ótimo para dados invisíveis é aquele com a maior margem $\frac{1}{\|\mathbf{w}\|^2}$, ou seja, de norma euclidiana mínima $\|\mathbf{w}\|^2$. Para um SVM linear, a melhor margem, que é o hiperplano ideal $\mathbf{w}$, é encontrada ao minimizar a função de custo nos dados de treinamento como mostra a Equação 2.9.

$$J(\mathbf{w}, \xi) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{K}\xi_i \tag{2.9}$$

ξ_i são as variáveis de folga, C é um parâmetro de regularização que controla a compensação entre a complexidade e o número de pontos não separáveis (GRAIMANN; ALLISON; PFURTSCHELLER, 2010). A Equação 2.9 está sujeita a Equação 2.10.

$$y_i(\mathbf{w}^T\mathbf{x}_i + w_0) \geq 1 - \xi_i \tag{2.10}$$

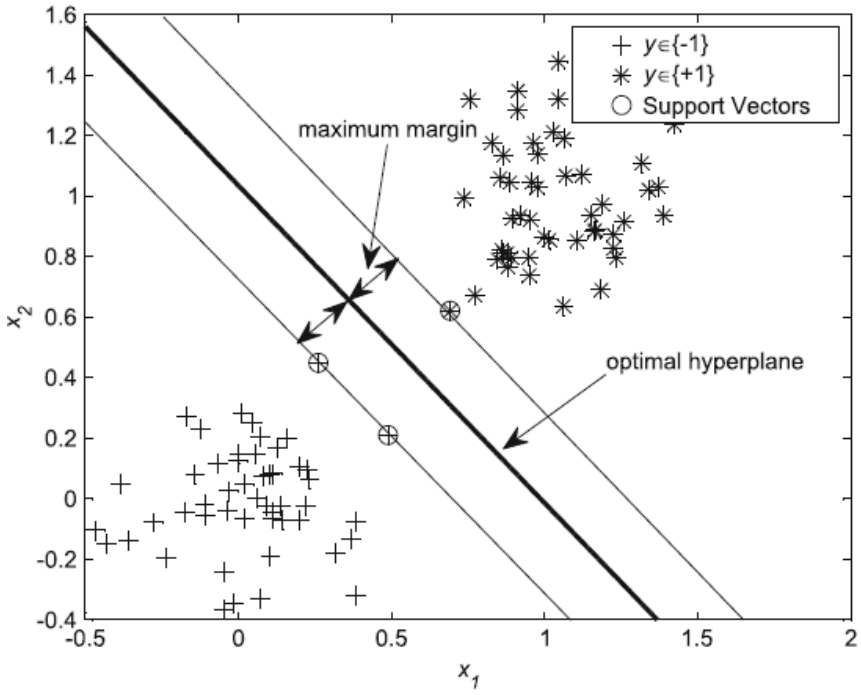


Figura 7 – Hiperplano ideal do treinamento de uma SVM. Fonte: Graimann, Allison e Pfurtscheller (2010)

onde $\xi_i \geq 0$, onde x_i denota o vetor de características de entrada i, K o número de entradas e $y_i \in \{-1, +1\}$ a associação de classe. As variáveis de folga medem o desvio dos pontos de dados da condição ideal de separabilidade do padrão (GRAIMANN; ALLISON; PFURTSCHELLER, 2010).

3 Materiais e Métodos

3.1 PocketBeagle

As plataformas da Fundação BeagleBoard.org são poderosos computadores de placa única (SBCs) que possuem um diferencial importante, pois foram criadas para serem um dispositivo para troca de informação devido ao seu pacote de microprocessador que contém dois microcontroladores adicionais no chip que podem ser usados para comunicação em tempo real (KRIDNER, 2017).

Ao contrário da maioria dos outros SBCs, as placas Beagle são totalmente de código aberto, podendo-se encontrar esquemas de fontes, layout de hardware, lista de materiais e manuais de referência técnica abrangentes na página da Fundação. Esse fácil acesso às especificações permite modificar o design da plataforma Beagle e integrá-la a um produto próprio, recurso útil para comercializar projetos (MOLLOY, 2019).

A PocketBeagle é uma plataforma embarcada de código aberto, de pequeno porte, sistema operacional Linux reduzido e com suporte da comunidade, também apresenta um incrível design de baixo custo e de fácil utilização, tornando-se uma placa de desenvolvimento ideal para iniciantes e profissionais com ambiente de desenvolvimento diretamente no navegador web. A disponibilidade de drivers de software de código aberto também permite a interface de dispositivos com a plataforma (KRIDNER, 2017).

A plataforma, que pode ser visualizada na Figura 8, é formada pela integração de um microprocessador de alto desempenho em uma placa de circuito impresso (PCB) e um extenso ecossistema de software. Apesar da impressionante capacidade dessa placa, ela não expõe completamente todos os recursos e interfaces do Sistema em Chip (SoC) Sitara AM335x da Texas Instruments (MOLLOY, 2019).

As PocketBeagle incorporam esse processador juntamente com memória *Double Data Rate 3* (DDR), gerenciamento de energia e todos os componentes passivos necessários em um encapsulamento *Ball-Grid Array*. Essa abordagem simplifica o layout do circuito da placa e permite seu tamanho reduzido (BeagleBoard.org Foundation, 2018).

A PocketBeagle é útil quando custo, tamanho e peso são considerados importantes para um projeto e apesar de não ter conectividade sem fio embarcada, isso pode ser modificado ao conectar módulos à porta serial universal (USB) ou ao transmissor/receptor assíncrono universal (UART) da placa, além do seu lado negativo não ter componentes para poder ser montado em uma PCB (MOLLOY, 2019).

Além disso, a PocketBeagle pode se conectar a periféricos USB usando o *USB On-the-Go* (OTG) que costuma ser utilizado em dispositivos que alternam entre as funções de cliente e host permitindo que o host PocketBeagle se conecte a um dispositivo escravo,

como um adaptador Wi-Fi ou Bluetooth (KRIDNER, 2017).

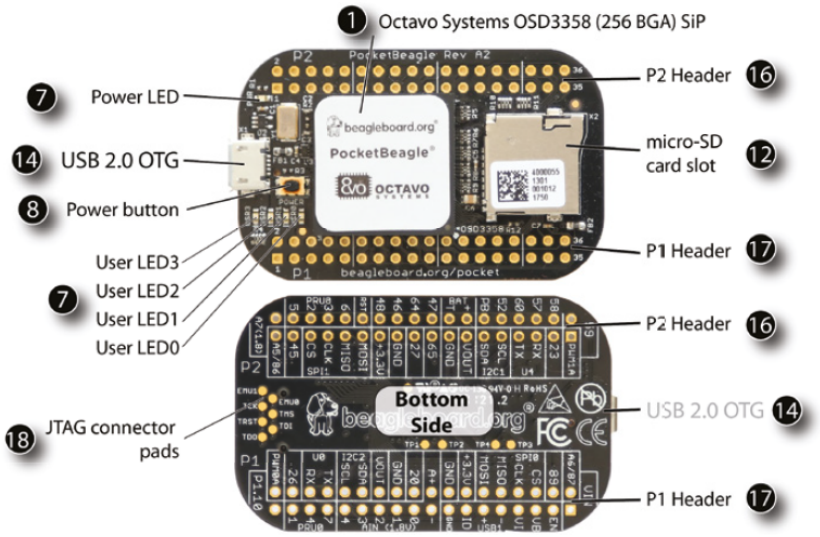


Figura 8 – Vista superior e inferior da plataforma PocketBeagle. Fonte: Molloy (2019)

A Figura 8 mostra os locais dos dispositivos, conectores, LEDs e interruptores no layout da placa PCB (MOLLOY, 2019). Algumas informações importantes são:

- O *Octavo Systems OSD3358-512M-BSM* é o sistema de processador da placa;
- Os *headers* P1 e P2 são despovoados para livre escolha de sua orientação;
- Os *User LEDs* são 4 LEDs azuis programáveis fornecidos ao usuário;
- O *Power button* pode ligar, desligar ou hibernar o sistema operacional;
- O *USB 2.0 OTG* funciona para conexão ao PC e para a alimentação da placa;
- O *Power LED* fornece comunicação sobre a energia da placa;
- No *micro-SD card slot* o cartão de memória com o sistema operacional será colocado.

A Figura 9 detalha os principais sistemas da PocketBeagle, onde o primeiro conjunto de textos explicativos, de 1 a 8, identifica e descreve os principais sistemas na plataforma (MOLLOY, 2019).

O microprocessador ARM Sitara AM335x Cortex A8 é um computador com um conjunto reduzido de instruções (RISC). Portanto, a 1.000 MHz, o processador executa 2.000 milhões de instruções por segundo e funciona a cerca de 1W inativo e 2,3 W para cargas pesadas de processamento (Texas Instruments Incorporated, 2017).

O próximo conjunto de textos explicativos, 9 a 19, identifica os vários conectores da plataforma, suas características físicas e suas funções, além do detalhamento das várias entradas e saídas que estão disponíveis nos *headers* de expansão (MOLLOY, 2019).

Existem 72 pinos na PocketBeagle (2x36), no entanto, nem todos estão disponíveis para entradas e saídas de uso geral (GPIOs). Os conectores restantes estão disponíveis

para serem multiplexados para muitas funções diferentes, várias das quais estão listadas na tabela (KRIDNER, 2017).

	Function	BeagleBone	PocketBeagle	Details
1	Processor	AM335x	OSD3358-SM	A powerful Texas Instruments Sitara ARM-A8 processor that is standalone or enclosed in an Octavo Systems System-In-Package (SiP) such as the OSD3358-SM.
		2 x PRUs	2 x PRUs	Programmable Real-time Units (PRUs). Microcontrollers that allow for real-time interfacing.
		Graphics Engine	Graphics Engine	Processor has a 3D graphics engine (Imagination Technologies PowerVR SGX530) that is capable of rendering 20 million polygons per second.
2	Graphics	HDMI Framer	None	The framer converts the LCD interface available on the AM335x processor into a HDMI signal (no HDCP).
3	Memory	512 MB DDR3	512 MB DDR3	The amount of system memory affects performance and the type of applications that can be run.
4	On-board Storage	eMMC (MMC1)	None	A 4GB on-board embedded multi-media card (eMMC), which is an SD card on a chip. The BeagleBone boards can boot without an SD card.
5	Power Management	TPS65217C	TPS65217C	Power management IC (PMIC). Sophisticated power management IC that has voltage regulators and is controlled by the main processor. Supports LiPo batteries.
6	Ethernet Processor	Ethernet PHY (10/100)	None	BBB can be connected to a network using a LAN8710A physical interface to an RJ45 connector. Not available on the wireless versions.
7	LEDs	6 x LEDs	4 x LEDs	Power LED and four user LEDs. The wired BeagleBone has LEDs on the RJ45 Ethernet socket (yellow = 100M link up, green = traffic).
8	Buttons	3 x Buttons	1 x Button	Power button. The BeagleBone boards have a reset button and a boot switch button for choosing to boot from the eMMC or the SD card.
	Connectors			
9	Video Out	micro-HDMI (HDMI-D)	None	For connecting to monitors and televisions. Supports resolutions up to 1280x1024 at 60Hz. It can run 1920x1080 but at 24Hz. Has HDMI CEC support.
		Audio out (HDMI-D)	None	HDMI can be broken out to a 3.5mm audio jack using accessories.
10	Network	Ethernet (RJ45)	None	10/100 Ethernet via a RJ45 connector. On board Wi-Fi and Bluetooth is available on the BeagleBone Black/Blue Wireless.
11	DC Power	5V DC supply (5.5mm) 12V DC supply on Blue	None	For connecting 5V DC mains PSUs to the board. PocketBeagle is usually powered via the USB connector but can be powered by battery or the expansion header.
12	SD Card	micro-SD card slot	micro-SD card slot	3.3V micro-SD card slot. The BeagleBone can be booted from this slot, flashed from this slot, or it can be used for additional storage when the board is booted from the eMMC.
13	Serial Debug	6 Pin Connector (0.1")	None	(UART0) Used with a serial TTL3V3 cable to connect to the serial console. This functionality is also available via USB on both boards.
14	USB	1 x USB 2.0 Client (mini-USB or micro-USB)	1 x USB OTG	(USB0) Connects to your desktop and can power the board.
15		1 x USB 2.0 Host (USB-A)		(USB1) You can connect USB peripherals (e.g., Wi-Fi) to the board with this connector.
16 17	Expansion Headers	Two 2x23 pin 0.1" female headers	Two 2x18 pin 0.1" unpopulated headers	These headers are multiplexed to provided access to a range of input/output features. Not all functionality is available at the same time. Used to connect capes.
18	Other Debug	JTAG	JTAG	Unpopulated JTAG header that can be used to debug a board when used with additional hardware and software.
19	Other Power	Battery connectors	via headers	It is possible to solder pins to these points on the BeagleBone or to the headers on the PocketBeagle to power the board. Read the SRM carefully!

Figura 9 – Subsistemas e conexões da plataforma PocketBeagle. Fonte: Molloy (2019)

3.2 Data Set IVa

Para realizar os testes e validações, serão utilizados nas implementações o *Data Set IVa* da *BCI Competition III* onde cada Data Set foi adquirido através de experimentos realizados pelos laboratórios líderes em tecnologia BCI e os arquivos divididos em duas partes: uma de treinamento e outra de teste (BLANKERTZ et al., 2006).

Os dados Data Set IVa foram adquiridos e armazenados utilizando amplificadores do tipo *BrainAmp* e uma touca com 128 canais de eletrodos de Ag/AgCl. 118 canais de EEG foram posicionados de acordo com o sistema 10/20, cada um destes canais foram filtrados em banda passante de 0,05Hz a 200Hz e depois digitalizados com uma frequência de amostragem de 1kHz com precisão de 16 bits. Também foram disponibilizados os mesmos dados com uma frequência de amostragem de 100Hz (DORNHEGE et al., 2004).

O Data Set foi adquirido de cinco sujeitos saudáveis e cada um sentou em uma cadeira confortável com os braços descansando nos apoios laterais da cadeira. O Data Set possui dados das quatro primeiras sessões e as sugestões visuais indicavam por 3,5s qual das três imagéticas motoras o sujeito deveria realizar: mão esquerda, mão direita ou pé direito. As sugestões visuais eram intercaladas por períodos aleatórios de 1,75s a 2,25s onde o sujeito podia relaxar (DORNHEGE et al., 2004).

A formatação do Data Set disponibiliza amostras de sinais contínuos de 118 canais de EEG e marcadores que indicam os pontos de tempo de 280 estímulos para cada um dos 5 sujeitos: A1, A2, A3, A4 e A5. Somente estímulos para as classes 'direito' e 'pé' são fornecidas no Data Set (DORNHEGE et al., 2004).

Os dados estão disponíveis no formato de arquivo Matlab (*.mat) que contém as variáveis:

- **cnt:** sinais contínuos de EEG, tamanho [tempo x canais];
- **mrk:** estrutura com as sugestões almejadas com os campos pos (vetor das posições das sugestões nos sinais de EEG), y (vetor das classes sugeridas) e className (vetor com o nome das classes);
- **info:** estrutura disponibilizando informações adicionais com os campos name (nome do Data Set), fs (taxa de amostragem), clab (indicação do canal), xpos (posição no eixo x do eletrodo em uma projeção 2D) e ypos (posição no eixo y do eletrodo em uma projeção 2D).

Também é disponibilizado no site da competição os dados no formato ASCII dividido em três conteúdos para cada sujeito:

- ***_cnt.txt:** sinal contínuo de EEG em que cada linha possui os valores para todos os canais em um momento específico;
- ***_mrk.txt:** informação da sugestão almejada em que cada linha representa uma sugestão onde o primeiro valor é o tempo que ocorreu e o segundo valor representa a classe;
- ***_nfo.txt:** contem outras informações que nem descrito no formato Matlab.

3.3 Validação dos resultados

O primeiro passo para a validação dos resultados desse trabalho será executar a mesma implementação desenvolvida por (LOTTE; GUAN, 2011) para identificar se os dados obtidos em suas pesquisas conferem com dados obtidos em outro computador.

O principal desafio ao realizar essa fase de testes será obter resultados precisos e reproduzíveis. Tempos de execução podem ser obtidos de várias maneiras, porém dependerá dos requisitos e das instalações disponíveis no computador onde será replicado o experimento.

Após comparar os parâmetros adquiridos e estabelecer relação com os resultados iniciais, será implementado um algoritmo para a plataforma PocketBeagle que realize a análise similar aos mesmos dados com classificador LDA.

Também será elaborado um algoritmo utilizando o classificador SVM para comparação com o algoritmo do classificador LDA elaborado anteriormente.

Serão utilizadas as medidas de desempenho descritas abaixo entre os dois classificadores para validação da plataforma como um possível sistema para implementações de interface cérebro-máquina.

3.3.1 Medidas de desempenho

Uma matriz de confusão é uma ferramenta de visualização normalmente usada para apresentar os resultados obtidos por um classificador. Cada coluna da matriz representa as instâncias em uma classe prevista, enquanto cada linha representa as instâncias em uma classe real (MONARD; BARANAUSKAS, 2003).

Com apenas duas classes, as escolhas estão estruturadas para predizer a ocorrência ou não de um evento simples (MONARD; BARANAUSKAS, 2003). Neste caso, os dois erros possíveis são denominados falso positivo (FP) e falso negativo (FN).

A Tabela 1 ilustrada uma matriz de confusão com duas classes, onde TP é o número de exemplos positivos classificados corretamente e TN é o número de exemplos negativos classificados corretamente de um total de n = (TP + FN + FP + TN) exemplos.

Um benefício da matriz de confusão é que ela facilita verificar se o classificador está confundindo duas classes ao rotular uma como outra (PANTIC et al., 2011).

A partir da matriz de confusão outras medidas podem ser derivadas, tais como confiabilidade, suporte, sensibilidade, especificidade, acurácia e cobertura (MONARD; BARANAUSKAS, 2003).

Tabela 1 – Modelo de uma matriz de confusão de duas classes. Fonte: Adaptado de Monard e Baranauskas (2003)

		Valor Predito	
		Classe 0	**Classe 1**
Valor Real	**Classe 0**	TP	FP
	Classe 1	FN	TN

Segundo Pantic et al. (2011), a forma mais óbvia de medir o desempenho de um classificador é calcular a taxa de classificação correta, ou seja, a acurácia, que é definida na Equação 3.1.

$$acurácia = \frac{TP + TN}{TP + FP + TN + FN} \tag{3.1}$$

Porém, em caso de dados desbalanceados, observar apenas a medida de acurácia pode induzir a uma conclusão equivocada quanto ao desempenho do classificador empregado, sendo que a classe majoritária pode encobrir o baixo desempenho da classe minoritária (SANTOS, 2017).

Ao utilizar a sensibilidade e a confiabilidade, que medem a qualidade de um processo de recuperação de informações, pode-se ter uma melhor análise do desempenho do classificador (PANTIC et al., 2011).

A sensibilidade descreve a integridade da recuperação, sendo definida como a parte dos exemplos positivos recuperados pelo processo versus o número total de exemplos positivos existentes como mostra a Equação 3.2 (PANTIC et al., 2011).

$$sensibilidade = \frac{TP}{TP + FN} \tag{3.2}$$

A confiabilidade descreve a precisão real da recuperação e é definida como a parte dos exemplos positivos que existem no número total de exemplos recuperados como mostra a Equação 3.3 (PANTIC et al., 2011).

$$confiabilidade = \frac{TP}{TP + FP} \tag{3.3}$$

Com essas duas medidas, pode-se gerar a medida chamada *f-score*, que é uma média ponderada entre confiabilidade e sensibilidade como mostra a Equação 3.4. Ao analisar essa medida, tem-se que um resultado mais próximo de 1 indica um melhor

desempenho do classificador e um resultado mais próximo de 0 demonstra desempenho ruim (SANTOS, 2017).

$$f - score = \frac{2 \times confiabilidade \times sensibilidade}{confiabilidade + sensibilidade} \tag{3.4}$$

Logo, para este trabalho, serão utilizadas as medidas de acurácia e *f-score* para validação dos classificadores desenvolvidos.

4 Resultados e Discussão

Neste capítulo são apresentados os resultados obtidos na execução do algoritmo de treinamento do classificador LDA desenvolvido por (LOTTE; GUAN, 2011) em Matlab sendo executado em computador pessoal no Linux, o programa em Python desenvolvido nesse trabalho no mesmo computador e na PocketBeagle.

4.1 Reprodução dos resultados

A reprodução dos resultados dos algoritmos apresentados no trabalho de (LOTTE; GUAN, 2011) é fundamental para comparação da implementação em Python e para o entendimento dos algoritmos em questão. A publicação de Lotte demonstra a classificação de sinais de EEG por algoritmo classificador baseado em CSP com LDA. Cada uma das variações do algoritmo foram reproduzidas em Matlab e estão descritas abaixo.

- **CSP:** algoritmo básico do extrator CSP utilizando matrizes de covariância inversas com otimização por multiplicadores de Lagrange;
- **DL CSP auto:** CSP regularizado pela metodologia sugerida em Ledoit-Wolf;
- **DL CSP:** CSP regularizado com Análise de Componentes Principais;
- **WTR CSP:** CSP com Regularização de Tikhonov normalizada;
- **TR CSP:** CSP com Regularização de Tikhonov.

Os resultados encontrados por (LOTTE; GUAN, 2011) em seu trabalho podem ser vistos na Tabela 2.

Tabela 2 – Resultados obtidos por Lotte e Guan em relação à acurácia da classificação. Fonte: elaborado a partir de Lotte e Guan (2011)

BCI Competition III Data Set IV						
Algoritmo	**A1**	**A2**	**A3**	**A4**	**A5**	**Média**
CSP	66,07%	96,43%	47,45%	71,88%	49,60%	66,29%
DL CSP auto	66,96%	96,43%	46,94%	71,43%	50%	66,35%
DL CSP	64,29%	96,43%	52,04%	71,88%	82,54%	73,44%
WTR CSP	69,64%	98,21%	54,59%	71,88%	85,32%	75,93%
TR CSP	71,43%	96,43%	63,27%	71,88%	86,90%	77,98%

Para realizar a execução dos códigos em Octave, utilizou-se laptop DELL com processador Intel Core i7-8550U CPU 4.0 GHz, SSD de 250GB, 8GB de RAM, placa de vídeo NVIDIA GeForce MX150 e sistema operacional Windows 10.

Para a reprodução dos algoritmos, foi utilizado o mesmo Data Set do artigo de Lotte e Guan (2011) e obteve-se os resultados em relação a precisão e tempo de treinamento para cada sujeito como mostra a Tabela 3.

Os resultados de acurácia encontrados nesta execução foram iguais ao encontrados por Lotte e Guan (2011), validando a reprodução do algoritmo.

No cálculo do tempo de treinamento não foi levado em consideração o tempo de carregamento dos arquivos do Data Set para a memória, apenas o tempo de treinamento do algoritmo e o treinamento do classificador.

Tabela 3 – Acurácia e tempos de treinamento dos algoritmos reproduzidos

Algoritmo	A1	A2	A3	A4	A5	Média
CSP	66,07 %	96,43 %	47,45 %	71,87 %	49,60 %	66,28 %
	215,36 ms	263,13 ms	117,96 ms	95,59 ms	74,49 ms	153,33 ms
DL CSP auto	66,96 %	96,43 %	46,94 %	71,43 %	50 %	66,35 %
	253,81 ms	309,63 ms	139,87 ms	99,12 ms	66,86 ms	173,86 ms
DL CSP	64,29 %	96,43 %	52,04 %	71,87 %	82,53 %	73,43 %
	20.138,47 ms	26.420,52 ms	11.412,30 ms	8.673,65 ms	5.502,43 ms	17.315,37 ms
WTR CSP	69,64 %	98,21 %	54,59 %	71,87 %	85,32 %	75,93 %
	22.920,65 ms	29.155,87 ms	14.600,89 ms	11.541,52 ms	8.993,45 ms	17.442,47 ms
TR CSP	71,43 %	96,43 %	63,27 %	71,87 %	86,91 %	77,98 %
	22.484,79 ms	28.955,95 ms	14.052,66 ms	11.047,95 ms	7.939,02 ms	16.896,07 ms

Cada tipo de BCI necessita de uma fase de calibração, na qual o sistema define os parâmetros para extrair as informações relevantes do sinal EEG, em termos de filtros e classificadores espaciais ou temporais (TURI; GAYRAUD; CLERC, 2020).

Essa fase é fundamental para o funcionamento da interface, porém, por normalmente demandar muito tempo do usuário, é considerada uma das limitações para a difusão do sistema BCI em aplicações reais (TURI; GAYRAUD; CLERC, 2020).

Dessa forma, deve-se buscar soluções que demandem o menor tempo possível de calibração do usuário, mesmo comprometendo em algum nível a acurácia da interface. Com este intuito, escolheu-se o filtro CSP como base para a implementação dos algoritmos em Python, por apresentar o menor tempo de treinamento na reprodução em Octave.

4.2 Algoritmo com filtro CSP e classificador LDA

4.2.1 Implementação

Apesar de um grande número de referências que descrevem a estrutura teórica dos BCIs baseados em imagens motoras, não há informações suficientes relacionadas ao software de computador disponível que possam ser adequados para desenvolver um BCI de propósito específico, eficiente e direto (ALONSO-VALERDI; SEPULVEDA, 2011), o que abre possibilidades de explorar as diferentes arquiteturas disponíveis.

A linguagem Python é uma das linguagens de programação mais acessíveis, pois possui sintaxe simplificada e compreensível. Devido à sua facilidade de aprendizado e uso, os códigos Python tem uma facilidade maior de escrita, além disso, essa linguagem tem um grande número de bibliotecas relacionadas a processamento digital de sinais, aprendizado de máquina e substituição de comandos do Octave.

O processamento de sinal EEG para aplicações BCI tem como objetivo traduzir os sinais EEG brutos para o estado mental estimado do usuário. Essa tradução é geralmente obtida usando uma abordagem de reconhecimento de padrões que necessitam principalmente de extração de recursos e classificação dos mesmos.

A extração é a primeira etapa de processamento que tenta encontrar nos sinais EEG algumas características relevantes do sinal para encontrar os estados mentais a serem identificados, enquanto rejeita o ruído e outras informações não relevantes. A classificação atribui uma classe a um conjunto de características extraídas dos sinais e essa classe corresponde ao tipo de estado mental identificado (LOTTE, 2014).

Para realizar a extração dos recursos, primeiramente necessitou-se compreender a estrutura do arquivo do Data Set disponibilizado em formato *.mat*. Assim, após importação do arquivo para um programa em linguagem Python, manipulou-se a estrutura fornecida para extrair as seguintes informações: número de amostras, número de canais, número de ensaios, taxa de amostragem, nome de cada canal, leitura de cada canal nos ensaios e leitura de cada canal nos testes.

Com esses dados em mãos, pode-se iniciar o processamento do sinal de EEG realizando a filtragem para retirar ruídos e informações irrelevantes. Como o Data Set foi gerado a partir de imagética motora, levou-se em consideração as bandas Mu e Beta para filtrar as ondas cerebrais registradas nos córtices somatossensorial e motor, por isso utilizou-se a banda passante entre 7Hz e 30Hz.

Realizou-se também o processo de filtragem espacial, pois, para sinais EEG de multicanais, os filtros espaciais são extremamente úteis para a análise de um ensaio janelado e para melhoria da relação sinal-ruído. Utilizou-se o CSP para essa filtragem devido a sua característica de obter uma distinção perceptiva nos estados mentais caracterizados pela dessincronização e sincronização relacionada aos eventos de imagética motora.

Após a implementação do algoritmo em Python e sua respectiva execução no mesmo computador definido anteriormente, foram obtidos os seguintes resultados de acurácia da classificação como mostra a Tabela 4.

Tabela 4 – Acurácia e tempo de treinamento do algoritmo com CSP e LDA

CSP/LDA Python	**A1**	**A2**	**A3**	**A4**	**A5**	**Média**
Acurácia	66,07%	96,43%	47,45%	71,88%	49,60%	66,29%
Tempo de Treinamento	1.099,39 ms	2.419,87 ms	442,81 ms	325,09 ms	143,61 ms	886,15 ms

Comparando os resultados de acurácia obtidos em Python com os resultados da reprodução em Octave, pode-se observar que os valores encontrados foram muito próximos, sendo em Octave uma acurácia de 66,28% e em Python de 66,29%.

Em relação ao tempo de treinamento da reprodução e da implementação, foram

realizadas 5 medições de tempo de treinamento para cada sujeito e os dados registrado na Tabela 4 são a média desses valores.

Comparando os tempos de treinamento do algoritmo CSP das Tabelas 3 e 4, pode-se notar que o programa em Octave teve uma melhor performance em relação ao programa em Python.

Antes de executar os programas, finalizou-se a execução de programas que estavam consumindo grande quantidade de CPU e memória do computador. A execução dos programas foi realizada na mesma máquina e no mesmo sistema operacional, pois qualquer divergência poderia acarretar em valores deturpados.

4.2.2 Análise de Perfil

Pode-se observar pela Tabela 4 que os tempos de treinamento da implementação em Python são maiores que os da reprodução e, devido a este fator, foi feita uma analise de perfil do código para identificar qual a função que consome mais memória e tempo de execução. Pode-se observar na Tabela 5 os valores encontrados para cada função executada.

As formas escolhidas para análise do perfil dos algoritmos deste trabalho foram a da utilização da CPU durante suas execuções e o tempo gasto executando as instruções das funções. Para realizar esta análise de perfil primeiro foi necessário preparar o sistema operacional da PocketBeagle, os detalhes dessa otimização podem ser observados no Apêndice A.

Realizar uma análise de perfil de um programa em Python é fazer uma análise dinâmica que mede as variáveis de interesse do programa e tudo o que as compõem.

Isso significa, no caso deste trabalho, medir o tempo gasto em cada uma de suas funções e fornecer dados sobre onde seu programa está gastando tempo e qual área vale a pena otimizar. Além de verificar quanto de memória foi gasta durante a execução e otimizar o código nos pontos mais críticos.

A análise de perfil de tempo de execução foi ordenada de forma decrescente de acordo com o tempo acumulado em cada função. Como o *LineProfiler* faz uma análise de todas as linhas do código, foram selecionadas as 5 funções que apresentaram maior tempo de execução no algoritmo de treinamento e os dados podem ser observados na Tabela 5.

A função *learnCSPLagrangian* é a função responsável pelo maior tempo de execução do código. Ela é responsável pela extração da matriz CSP dos sujeitos e a que consome mais tempo de execução. A segunda maior é a função responsável pela filtragem dos sinais de EEG que é chamada 10 vezes já que necessita filtrar os sinais de treinamento e teste de cada um dos 5 sujeitos.

A função *scipy.io.loadmat*, mesmo sendo chamada poucas vezes, possui um alto

Tabela 5 – Análise de perfil das funções de treinamento implementadas

Número de Chamadas	**Tempo Acumulado**	**Tempo por Chamada**	**Porcentagem do Tempo de Execução**	**Função**
5	4.211,38 ms	842,28 ms	47,2%	learnCSPLagrangian
10	2.768,99 ms	276,90 ms	31,0%	eegButterFilter
2	1.138,46 ms	569,23 ms	12,8%	scipy.io.loadmat
10	720,86 ms	72,09 ms	10,2%	extractFeatures
5	15,92 ms	3,18 ms	0,2%	LDATrain
5	6,07 ms	1,21 ms	0,1%	LDATest

tempo de execução. Isso acontece porque ela é a responsável por carregar os sinais de EEG do arquivo para o programa

Já a função *extractCSPFeatures* tem o quarto maior tempo de execução quando olha-se o tempo acumulado, porém ao analisar o tempo por chamada e a quantidade de vezes que ela é chamada, percebe-se que ela não consome tanto tempo de execução do programa como um todo.

Com isso, percebe-se que extrair os atributos com o CSP demanda mais tempo de processamento do que a classificação com o LDA.

Um dos recursos menos conhecidos do pacote *MemoryProfiler* é sua capacidade de representar graficamente o consumo de memória em função do tempo. Além disso, é possível ver com marcações de colchetes coloridos em que momento as funções monitoradas iniciaram e terminam suas execuções.

Pode-se observar na Figura 10 o gráfico do consumo de memória em função do tempo do código executado em um computador pessoal.

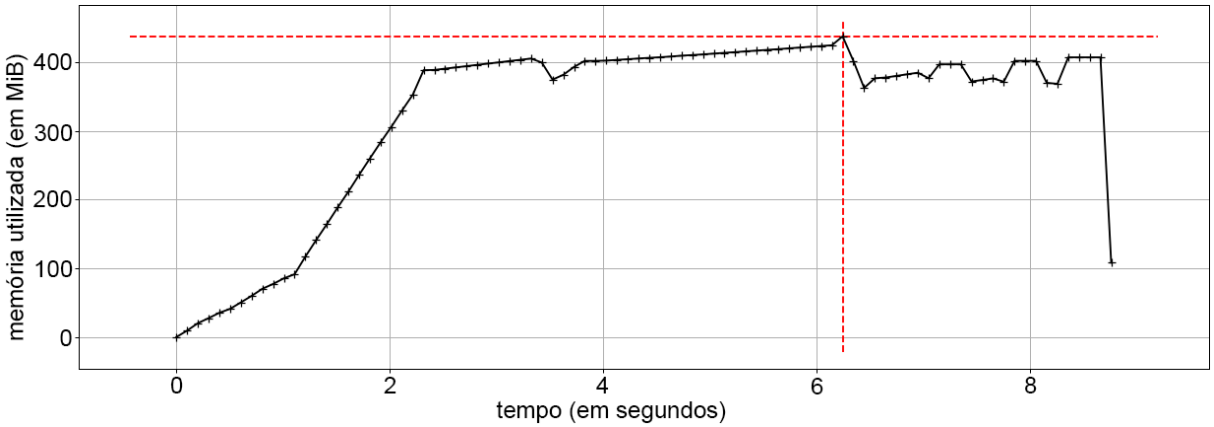


Figura 10 – Consumo de memória em função do tempo no computador pessoal

Ao analisar que o consumo de memória era constante ao longo da execução do algoritmo, percebeu-se que a maior utilização acontecia no carregamento dos Data Sets de cada sujeito.

Para resolver esse problema, optou-se por modificar o código e carregar o Data Set

de cada sujeito a medida que ele fosse necessário para a execução. Com isso, gerou-se o gráfico de consumo de memória apresentado na Figura 11.

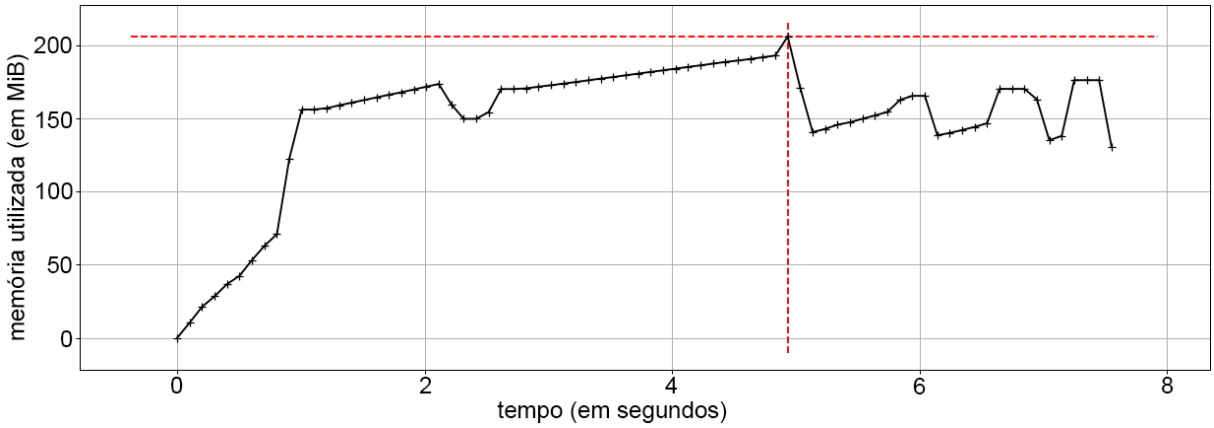


Figura 11 – Consumo de memória do algoritmo com LDA no computador pessoal com os EEGs dos sujeitos separados

Pode-se notar que, com a modificação do carregamento dos Data Sets, o pico de consumo de memória que antes era de 460MB diminuiu para 216MB, uma redução 53,07%. Em relação ao tempo de execução do código, diminuiu de 8,75s para 7,55s, uma redução de 13,71%.

Ao realizar a análise de perfil de tempo de execução novamente e selecionar as 6 funções principais que apresentaram maior tempo de execução no algoritmo de treinamento, obteve-se os dados apresentados na Tabela 6.

Tabela 6 – Análise de perfil das funções de treinamento implementadas

Número de Chamadas	**Tempo Acumulado**	**Tempo por Chamada**	**Porcentagem do Tempo de Execução**	**Função**
5	4.241,89 ms	848,38 ms	52,4%	learnCSPLagrangian
10	2.523,73 ms	252,37 ms	31,2%	eegButterFilter
10	657,82 ms	65,78 ms	8,0%	extractCSPFeatures
5	342,66 ms	68,53 ms	4,2%	scipy.io.loadmat
5	21,82 ms	4,36 ms	0,3%	LDATrain
5	5,52 ms	1,10 ms	0,1%	LDATest

Com a mudança do carregamento dos arquivos do Data Set, a função *scipy.io.loadmat* foi chamada mais vezes, porém gastou menos tempo em execução e, como consequência, diminuiu sua porcentagem no total do tempo de execução do programa.

4.2.3 Matriz de Confusão

Pode-se observar nas Tabelas 7 a 11 as matrizes de confusão de cada sujeito geradas pela implementação do algoritmo com CSP e LDA para as classes ‘direito’ e ‘pé’, que correspondem a classe 0 e 1 respectivamente.

Tabela 7 – Matriz de confusão do sujeito A1

Sujeito A1		**Valor Predito**	
		Classe 0	**Classe 1**
Valor Real	**Classe 0**	35	13
	Classe 1	25	39

Tabela 8 – Matriz de confusão do sujeito A2

Sujeito A2		**Valor Predito**	
		Classe 0	**Classe 1**
Valor Real	**Classe 0**	28	2
	Classe 1	0	26

Tabela 9 – Matriz de confusão do sujeito A3

Sujeito A3		**Valor Predito**	
		Classe 0	**Classe 1**
Valor Real	**Classe 0**	75	80
	Classe 1	23	18

Tabela 10 – Matriz de confusão do sujeito A4

Sujeito A4		**Valor Predito**	
		Classe 0	**Classe 1**
Valor Real	**Classe 0**	53	6
	Classe 1	57	108

Tabela 11 – Matriz de confusão do sujeito A5

Sujeito A5		**Valor Predito**	
		Classe 0	**Classe 1**
Valor Real	**Classe 0**	111	116
	Classe 1	11	14

A partir das matrizes de confusão do classificador LDA, calculou-se os valores de acurácia, confiabilidade, sensibilidade e *f-score* de cada sujeito para gerar a Tabela 12. Calculou-se também a média do classificador.

Ao analisar as medidas de desempenho do classificador LDA desenvolvido, percebe-se uma performance mediana para a maioria dos sujeitos do Data Set, o que gera uma média de acurácia de 66,29% e um *f-score* de 0,6940.

Tabela 12 – Medidas de desempenho do classificador LDA

Sujeitos	**A1**	**A2**	**A3**	**A4**	**A5**	**Média**
Acurácia	66,07%	96,43%	47,45%	71,88%	49,60%	66,29%
Confiabilidade	72,92%	93,33%	48,39%	89,83%	48,90%	70,67%
Sensibilidade	58,33%	100,00%	76,53%	48,18%	90,98%	74,81%
f-score	0,6481	0,9655	0,5929	0,6272	0,6361	0,6940

4.3 Algoritmo com filtro CSP e classificador SVM

4.3.1 Implementação

Para a implementação do código com o classificar SVM, utilizou-se como base o mesmo código elaborado com o classificador LDA e com os Data Sets separados. Retirou-se as funções de análise discriminante e inseriu-se as funções de máquina de vetores de suporte, mas a estrutura do código permaneceu semelhante.

Após a elaboração do algoritmo em Python, executou-se no mesmo no computador definido anteriormente e foram obtidos os resultados de acurácia e tempo de treinamento apresentados na Tabela 13.

Ao comparar as implementações com SVM e com LDA, nota-se que o pico de consumo de memória de ambas é igual, porém houve uma redução de 1,08% em relação ao tempo de execução com o novo classificador.

Tabela 13 – Acurácia e tempo de treinamento do algoritmo com CSP e SVM

CSP/SVM Python	**A1**	**A2**	**A3**	**A4**	**A5**	**Média**
Acurácia	98,21%	100,00%	77,55%	80,80%	59,13%	83,14%
Tempo de Treinamento	1.078,48 ms	2.385,79 ms	438,83 ms	337,05 ms	142,57 ms	876,54 ms

4.3.2 Análise de Perfil

Da mesma forma que foi realizada no algoritmo com LDA, analisou-se quais funções consumiam mais tempo no algoritmo com SVM. A Tabela 14 mostra as 6 funções principais do algoritmo e suas proporções no tempo de execução.

Tabela 14 – Análise de perfil das funções de treinamento implementadas com SVM

Número de Chamadas	**Tempo Acumulado**	**Tempo por Chamada**	**Porcentagem do Tempo de Execução**	**Função**
5	4.471,95 ms	894,39 ms	57,5%	learnCSPLagrangian
10	2.298,47 ms	229,84 ms	29,5%	eegButterFilter
10	648,45 ms	64,84 ms	8,4%	extractFeatures
5	324,17 ms	64,83 ms	4,2%	scipy.io.loadmat
5	6,68 ms	1,34 ms	0,1%	extractSVMmodel
5	7,55 ms	1,51 ms	0,1%	accuracySVM

Assim como no algoritmo com LDA, o maior consumidor do tempo de execução foi a função que calcula a matriz CSP e as 4 funções que consomem mais tempo permaneceram iguais.

Porém ao analisar as funções dos classificadores, as funções do SVM consumiram um total de 14,23 ms e as do LDA 27,34 ms. Essa diferença mostra que é mais rápido classificar com SVM do que com LDA no computador pessoal.

A Figura 12 mostra o consumo de memória do algoritmo com SVM. Comparando a implementação com SVM em relação a com LDA, percebe-se que o uso de memória e tempo de execução do código foram muito similares, pois a execução com SVM atingiu o pico de 216MB de memória utilizada em um tempo de execução de 7,54s.

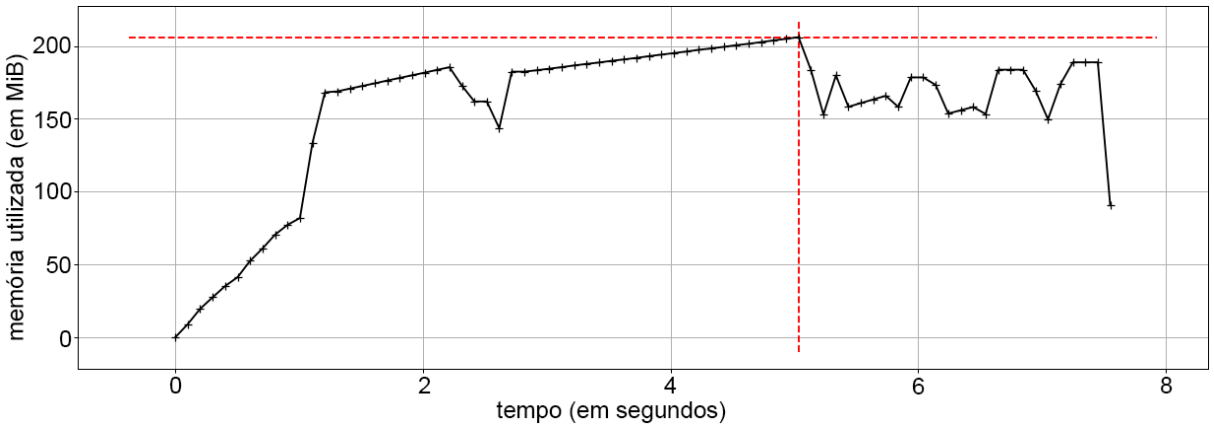


Figura 12 – Consumo de memória do algoritmo com SVM no computador pessoal com os EEGs dos sujeitos separados

4.3.3 Matriz de Confusão

Pode-se observar nas Tabelas 15 a 19 as matrizes de confusão de cada sujeito geradas pela implementação do algoritmo com CSP e SVM.

Tabela 15 – Matriz de confusão do sujeito A1

Sujeito A1		**Valor Predito**	
		Classe 0	**Classe 1**
Valor Real	**Classe 0**	60	0
	Classe 1	2	50

Tabela 16 – Matriz de confusão do sujeito A2

Sujeito A2		**Valor Predito**	
		Classe 0	**Classe 1**
Valor Real	**Classe 0**	28	0
	Classe 1	0	28

Tabela 17 – Matriz de confusão do sujeito A3

Sujeito A3		**Valor Predito**	
		Classe 0	**Classe 1**
Valor Real	**Classe 0**	67	31
	Classe 1	13	85

Tabela 18 – Matriz de confusão do sujeito A4

Sujeito A4		**Valor Predito**	
		Classe 0	**Classe 1**
Valor Real	**Classe 0**	83	27
	Classe 1	16	98

Tabela 19 – Matriz de confusão do sujeito A5

Sujeito A5		**Valor Predito**	
		Classe 0	**Classe 1**
Valor Real	**Classe 0**	105	17
	Classe 1	86	44

A partir das matrizes de confusão do classificador SVM, gerou-se a Tabela 20 que apresenta os dados em relação às medidas de desempenho deste classificador para cada sujeito e a média entre eles.

Ao analisar essas medidas do classificador SVM desenvolvido, percebe-se uma alta performance para a maioria dos sujeitos do Data Set, o que gera uma média de acurácia de 83,14% e um *f-score* de 0,8111.

Ao comparar os *f-score* dos classificadores, percebe-se que o do SVM teve um melhor desempenho nas classificações do que o do LDA que teve 0,6940, ou seja, uma diferença de 14,44%.

Tabela 20 – Medidas de desempenho do classificador SVM

Sujeitos	**A1**	**A2**	**A3**	**A4**	**A5**	**Média**
Acurácia	98,21%	100,00%	77,55%	80,80%	59,13%	83,14%
Confiabilidade	100,00%	100,00%	73,28%	78,40%	72,13%	84,76%
Sensibilidade	96,15%	100,00%	86,73%	85,96%	33,85%	80,94%
f-score	0,9804	1,0000	0,7944	0,8201	0,4607	0,8111

4.4 Comparativo dos Resultados na PocketBeagle

Após a execução dos códigos no computador pessoal, executou-se os algoritmos implementados com LDA e SVM com os Data Set separados na PocketBeagle para verificar o consumo de memória e os tempos de treinamento para cada sujeito.

Na Tabela 21 pode-se observar um comparativo dos tempos de treinamento obtidos nas execuções de ambos os algoritmos na PocketBeagle.

Em relação ao tempo de treinamento de cada sujeito, houve mais diminuições do que aumentos e as reduções que aconteceram tiveram proporções maiores em comparação com os acréscimos. Pode-se constatar uma diminuição de 0,74% na média do tempo de treinamento do algoritmo com SVM em relação ao do com LDA.

Tabela 21 – Comparação do tempo de treinamento dos algoritmos na PocketBeagle

Sujeitos	**A1**	**A2**	**A3**	**A4**	**A5**	**Média**
LDA	62,25 s	94,67 s	29,34 s	19,07 s	9,56 s	42,98 s
SVM	62,38 s	93,03 s	29,38 s	19,02 s	9,47 s	42,66 s
Comparação	+0,21%	-1,73%	+0,14%	-0,26%	-0,94%	-0,74%

Na Tabela 22 pode-se observar um comparativo das acurácias obtidas nas execuções de ambos os algoritmos na PocketBeagle.

Tabela 22 – Comparação da acurácia dos algoritmos na PocketBeagle

Sujeitos	**A1**	**A2**	**A3**	**A4**	**A5**	**Média**
LDA	66,07%	96,43%	47,45%	71,88%	49,60%	66,29%
SVM	98,21%	100,00%	77,55%	80,80%	59,13%	83,14%
Comparação	+48,65%	+3,70%	+63,43%	+12,41%	+19,21%	+25,42%

O algoritmo com SVM demonstrou um aumento na acurácia em todos os sujeitos ao comparar com o algoritmo com LDA. Pode-se notar que todos os sujeitos ficaram com acurácia acima dos 59% com o algoritmo implementado com SVM e com uma média de 83,14% no geral.

Na Tabela 23 pode-se observar um comparativo dos *f-score* obtidas nas execuções de ambos os algoritmos na PocketBeagle. O classificador SVM teve um aumento quando comparado com o classificador LDA em todos os sujeitos com exceção de um. Observando a média geral, o classificador SVM teve um aumento de 16,87% em relação ao LDA.

Tabela 23 – Comparação do *f-score* dos algoritmos na PocketBeagle

Sujeitos	**A1**	**A2**	**A3**	**A4**	**A5**	**Média**
LDA	0,6481	0,9655	0,5929	0,6272	0,6361	0,6940
SVM	0,9804	1,0000	0,7944	0,8201	0,4607	0,8111
Comparação	+51,27%	+3,57%	+33,99%	+30,76%	-29,15%	+16,87%

Em relação ao consumo de memória da PocketBeagle, com o algoritmo com LDA otimizado, executou-se o código na PocketBeagle e gerou-se o gráfico apresentado na Figura 13 e com o algoritmo com SVM, gerou-se o gráfico apresentado na Figura 14.

Observa-se que o tempo de execução dos algoritmos na PocketBeagle foram bem mais elevados que no computador pessoal, porém não apresentou qualquer diferença na acurácia dos algoritmos. Ao comparar as execuções no sistema embarcado, ambas tiveram valores muito próximos, a execução do algoritmo com LDA durou 288,5s e a com SVM durou 290,6s.

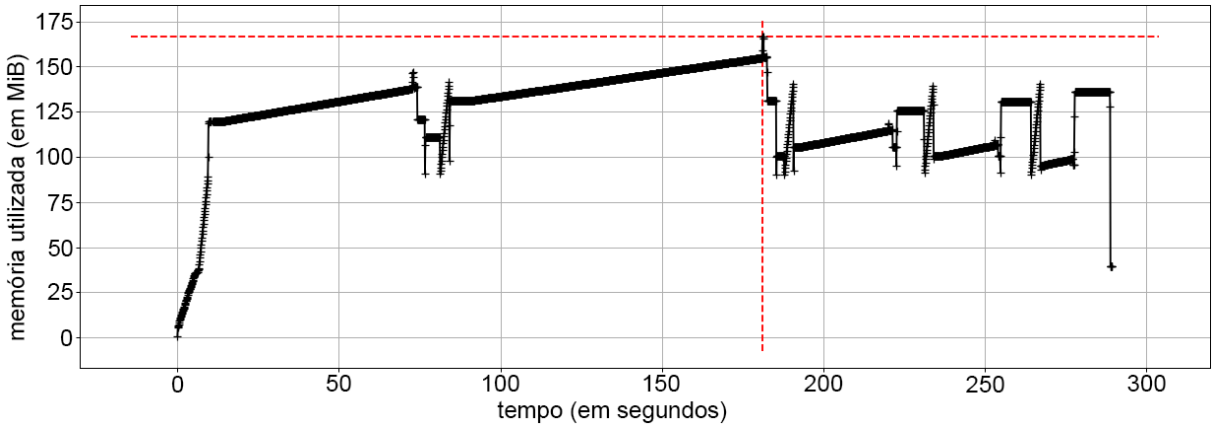


Figura 13 – Consumo de memória em função do tempo do LDA na PocketBeagle

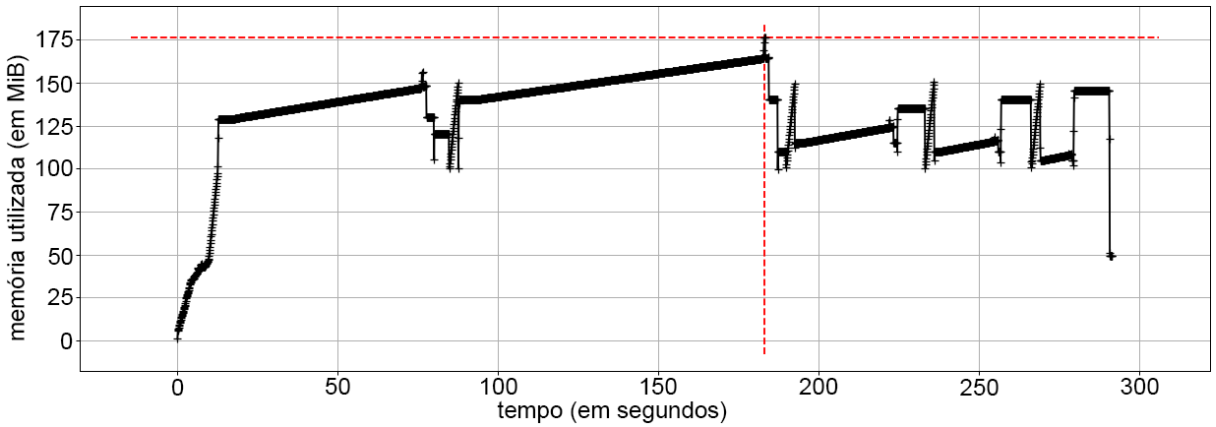


Figura 14 – Consumo de memória em função do tempo do SVM na PocketBeagle

Esse aumento no tempo de execução era esperado, tendo em vista que o computador pessoal possui maior capacidade computacional que o sistema embarcado. Porém o consumo de memória foi menor na PocketBeagle em ambas as implementações do que nas execuções no computador pessoal. No sistema embarcado, o algoritmo LDA atingiu um pico de 175MB enquanto o com SVM atingiu 185MB.

Para verificar o consumo de energia da PocketBeagle utilizou-se um *USB tester* que calcula a tensão e corrente de dispositivos conectados a ele.

O *USB tester* pode medir uma tensão de 4V a 20V com uma resolução de 10mV e uma corrente de 0 a 3A com uma resolução de 10mA. A faixa de erro da tensão é de $\pm 1\%$ e da corrente de $\pm 2\%$.

Utilizando essa informação, calculou-se a potência do sistema embarcado em três momentos diferentes: em estado ocioso, executando o algoritmo LDA e executando o algoritmo SVM. Os valores encontrados podem ser observados na Tabela 24.

Tabela 24 – Consumo de energia da PocketBeagle

Medidas	**Tensão**	**Corrente**	**Potência**
Ocioso	4,82 V	3,11 mA	15,0 mW
LDA	4,81 V	4,60 mA	22,1 mW
SVM	4,84 V	4,60 mA	22,3 mW

Ao subtrair o consumo de energia da PocketBeagle enquanto executava os algoritmos pelo consumo da plataforma ociosa, verifica-se que o classificador LDA em sua execução utilizou 7,1mW e o classificador SVM utilizou 7,3mW.

Pode-se verificar por esses dados que os algoritmos tiveram um consumo de energia muito próximo um do outro, sendo que o LDA teve um consumo um pouco melhor.

4.5 Seção de Contribuição

Este trabalho expõe questões de desempenho relacionadas à implementação em sistemas embarcados de algoritmos para BCI e suas principais contribuições são:

- A apresentação de dois modelos de classificação de sinais de EEG de imagética motora que oferecem suporte à compreensão do desempenho de BCIs. Estes modelos de classificação tornam-se a base desta pesquisa, que é usada como uma ferramenta para apoiar a análise de desempenho e design de algoritmos.
- A disponibilização de um conjunto de metodologias de referência para quantificar os parâmetros de desempenho de modelos de classificação. Estes modelos de classificação implementados e seus resultados constituem uma ferramenta de avaliação e análise de desempenho.
- Para estudar o consumo de memória dos classificadores, explorou-se como o consumo pode afetar o desempenho final do ponto de vista estrutural. Mostrou-se como o carregamento dos sinais de EEG ao início do programa e individualmente quando necessário interagem com o tempo de execução dos algoritmos e o alcance do pico do uso de memória.

5 Conclusões

Este trabalho apresentou um estudo da implementação em sistema embarcado de algoritmos de treinamento do classificador LDA e do classificador SVM, comparando os resultados obtidos utilizando linguagens diferentes, sendo elas as implementações em Python e Octave, elaborada por Lotte e Guan (2011), e a mesma arquitetura para as implementações em Python, utilizando a PocketBeagle.

Além disso, foi realizado um levantamento histórico dos principais acontecimentos da área de BCI e conceitos como extração de sinais de EEG, processamento desses sinais e extração de características para classificação.

Uma das propostas deste trabalho era de obter um tempo de treinamento na implementação em Python próximo ou menor que na implementação em Octave, entretanto os resultados apresentados mostram que a implementação em Python necessitou, em média, de quase 6 vezes mais do que o tempo de treinamento da implementação em Octave.

Ao embarcar as implementações em Python na PocketBeagle, esse tempo de treinamento aumentou para, em média, 42,98s com o classificador LDA e 42,66s com o classificador SVM. Ao comparar com a execução em computador pessoal, o tempo de treinamento do classificador LDA aumentou em 48,50 vezes e do classificador SVM em 48,14 vezes.

Devido a esse aumento no tempo de treinamento da PocketBeagle, não seria recomendado realizar treinamentos com os algoritmos desenvolvidos para este trabalho em tempo real no sistema embarcado. Porém, treinar um modelo com esses algoritmos e utilizar o modelo já treinado em tempo real no sistema embarcado pode ser uma possibilidade de uso.

Outro objetivo deste trabalho era analisar os consumos de memória das implementações no sistema embarcado. Ao observar os resultados apresentados, observa-se que as interfaces implementadas consumiram menos da metade da memória disponível na PocketBeagle de 512MB, sendo o consumo do classificador LDA de 175MB em seu pico e do SVM de 185MB no pico da execução.

O desempenho destes algoritmos foi avaliado a partir de métricas decorrentes da matriz de confusão, que registra em suas linhas e colunas os erros e acertos da predição e gera medidas como acurácia, confiabilidade, sensibilidade e *f-score* como métricas de validação.

Fica claro, ao utilizar o mesmo Data Set, que o classificador SVM tem melhor desempenho do que o LDA, pois sua acurácia é de 83,14% e seu *f-score* é 0,8111, enquanto para o classificador LDA têm-se acurácia de 66,29% e *f-score* de 0,6940.

Observando o consumo de energia dos algoritmos, percebe-se que ambos possuem um consumo parecido, sendo o consumo do classificador LDA de 7,1mV e do classificador SVM de 7,3mV. Quando comparado ao consumo de energia da placa em estado ocioso, o LDA e o SVM tiveram um aumento de 47,33%.

Este trabalho mostrou que é possível aplicar métodos de processamento em arquitetura embarcada para classificação de sinais de imagética motora. Portanto, para trabalhos futuros, podem ser utilizadas diferentes plataformas para comparar seus desempenhos com os encontrados neste estudo.

Também fica como sugestão a análise se a presença do filtro espacial interfere na taxa de acerto dos classificadores, visto que sua utilização é responsável por mais da metade do tempo de treinamento dos algoritmos implementados.

Outra possibilidade de trabalho futuro seria a refatoração do código visando um menor consumo de CPU, uma vez que com pequenos ajustes no carregamento dos sinais de EEG dos sujeitos já foi possível melhorar o consumo no sistema embarcado. Pode-se também realizar uma otimização dos algoritmos ao alterar sua arquitetura para obter um melhor tempo de treinamento ao utilizar linguagens como Rust, Cython ou C.

Referências

ALONSO-VALERDI, L. M.; SEPULVEDA, F. Python in brain-computer interfaces (bci): Development of a bci based on motor imagery. In: . [S.l.: s.n.], 2011. p. 74–79. Citado na página 52.

BeagleBoard.org Foundation. *Site oficial da Beagle Board*. 2018. Disponível em: <https://beagleboard.org>. Acesso em: 13 de abril de 2019. Citado na página 44.

BEAR, M. F.; CONNORS, B. W.; PARADISO, M. A. *Neuroscience: Exploring the Brain*. ISBN 9780781760034: Lippincott Williams Wilkins, 2007. Citado na página 29.

BELWAFI, K. et al. An embedded implementation based on adaptive filter bank for brain–computer interface systems. *Journal of Neuroscience Methods*, v. 305, p. 1–16, 2018. Citado na página 26.

BLANKERTZ, B. et al. The bci competition iii: Validating alternative approaches to actual bci problems. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, v. 14, n. 2, p. 153–159, 2006. Citado na página 47.

Blankertz, B. et al. Optimizing spatial filters for robust eeg single-trial analysis. *IEEE Signal Processing Magazine*, v. 25, n. 1, p. 41–56, 2008. Citado 2 vezes nas páginas 39 e 40.

BRAIN Initiative. *Why study the brain?* 2016. Disponível em: <https://www.braininitiative.org/why-study-the-brain/>. Acesso em: 26 de janeiro de 2019. Citado na página 23.

CARTER, R. *The Human Brain Book: An Illustrated Guide to its Structure, Function, and Disorders*. ISBN 9781465431622: DK Publishing, 2014. Citado 4 vezes nas páginas 23, 24, 28 e 29.

CLARK, D.; BOUTROS, N.; MENDEZ, M. *The Brain and Behavior: An Introduction to Behavioral Neuroanatomy*. ISBN 9780521840507: Cambridge University Press, 2005. Citado 2 vezes nas páginas 28 e 29.

CLERC, M.; BOUGRAIN, L.; LOTTE, F. *Brain-Computer Interfaces 2*. ISBN 9781848219632: Wiley, 2016. Citado 4 vezes nas páginas 36, 37, 40 e 42.

CRONE, N. E. et al. Functional mapping of human sensorimotor cortex with electrocorticographic spectral analysis. i. alpha and beta event- related desynchronization. *Brain: a journal of neurology*, v. 121, n. 12, p. 2271–2299, 1999. Citado na página 24.

DALY, J. J.; HUGGINS, J. E. Brain–computer interface: Current and emerging rehabilitation applications. *Archives of Physical Medicine and Rehabilitation*, v. 96, n. 3, p. S1–S7, 2015. Citado na página 26.

DORNHEGE, G. et al. Boosting bit rates in non-invasive eeg single-trial classifications by feature combination and multi-class paradigms. *IEEE Transactions on Biomedical Engineering*, v. 51, n. 6, p. 993–1002, 2004. Citado na página 47.

GRAIMANN, B.; ALLISON, B.; PFURTSCHELLER, G. *Brain-Computer Interfaces: Revolutionizing Human–Computer Interaction*. ISBN 9783642020919: Springer, 2010. Citado 10 vezes nas páginas 13, 25, 31, 32, 33, 34, 38, 39, 42 e 43.

GUNASEKERA, B. et al. Intracortical recording interfaces: Current challenges to chronic recording function. *ACS Chemical Neuroscience*, v. 6, n. 1, p. 68–83, 2015. Citado na página 31.

HAMALEINEN, M. et al. Magnetoencephalography theory, instrumentation, and applications to noninvasive studies of the working human brain. *Reviews of Modern Physics*, v. 65, p. 413–497, 1993. Citado na página 34.

HAZARIKA, N. et al. Classification of eeg signals using the wavelet transform. *Signal Process*, v. 59, n. 1, p. 61–72, 1997. Citado na página 32.

KAVITHA, A.; KRISHNAVENI, V. Eeg signal analysis for detection of abnormalities with cwt and svm technique. *Australian Journal of Basic and Applied Sciences*, v. 9, n. 20, p. 442–449, 2015. Citado na página 34.

KRIDNER, J. *Pocket Beagle System Reference Manual*. 2017. Disponível em: <https://github.com/beagleboard/pocketbeagle/wiki/System-Reference-Manual>. Acesso em: 18 de maio de 2019. Citado 3 vezes nas páginas 44, 45 e 46.

LARSEN, E. A. *Classification of EEG signals in a brain-computer interface system*. Dissertação (Mestrado) — Universidade Norueguesa de Ciência e Tecnologia, Department of Computer and Information Science, Computer Science. Trondheim, Junho 2007. Citado 2 vezes nas páginas 25 e 31.

LOTTE, F. *A Tutorial on EEG Signal Processing Techniques for Mental State Recognition in Brain-Computer Interfaces*. HAL 01055103: Springer, 2014. Citado na página 53.

LOTTE, F.; GUAN, C. Regularizing common spatial patterns to improve bci designs: Unified theory and new algorithms. *IEEE Transactions on Biomedical Engineering*, v. 58, n. 2, p. 355–362, 2011. Citado 5 vezes nas páginas 15, 27, 48, 51 e 64.

LOTTE, F. et al. A review of classification algorithms for eeg-based brain-computer interfaces. *Journal of Neural Engineering*, v. 4, n. 2, p. R1–R13, 2007. Citado na página 25.

microwavemont by Tindie. *OLED with 4-port USB HUB cape for PocketBeagle*. 2020. Disponível em: <https://www.tindie.com/products/microwavemont/oled-with-24-port-usb-hub-cape-for-pocketbeagle/>. Acesso em: 03 de dezembro de 2020. Citado na página 72.

MOKIENKO, O. et al. Motor imagery and its practical application. *Neuroscience and Behavioral Physiology*, v. 44, n. 5, p. 483–489, 2014. Citado na página 25.

MOLLOY, D. *Exploring BeagleBone: Tools and Techniques for Building with Embedded Linux*. ISBN 9781118935125: Wiley, 2019. Citado 4 vezes nas páginas 13, 44, 45 e 46.

MONARD, M. C.; BARANAUSKAS, J. A. Conceitos sobre aprendizado de máquina. In: *Sistemas Inteligentes Fundamentos e Aplicações*. 1. ed. Barueri-SP: Manole Ltda, 2003. p. 89–114. ISBN 85-204-168. Citado 3 vezes nas páginas 15, 48 e 49.

Montessori Public. *Penfield Homunculus*. 2016. Disponível em: <https://www.montessoripublic.org/2016/08/penfield-montessori-academy-new-chapter-milwaukee/penfield-homunculus/>. Acesso em: 08 de fevereiro de 2019. Citado 2 vezes nas páginas 13 e 30.

MULDER, T. Motor imagery and action observation: cognitive tools for rehabilitation. *Journal of Neural Transmission*, v. 114, n. 10, p. 1265–1278, 2007. Citado 2 vezes nas páginas 24 e 25.

NAM, C. S.; NIJHOLT, A.; LOTTE, F. *Brain–Computer Interfaces Handbook: Technological and Theoretical Advances*. ISBN 9781498773430: CRC Press, 2018. Citado 9 vezes nas páginas 13, 29, 31, 32, 33, 37, 38, 39 e 41.

NICOLAS-ALONSO, L.; GOMEZ-GIL, J. Brain computer interfaces, a review. *Sensors 2012*, v. 12, n. 2, p. 1211–1279, 2012. Citado na página 38.

NIEDERMEYER, E.; DA SILVA, F. L. *Electroencephalography: Basic Principles, Clinical Applications and Related Fields*. ISBN 9780781751261: Lippincott Williams Wilkins, 2005. Citado na página 37.

PANTIC, M. et al. *Computer Based Coursework Manual Machine Learning (Course 395)*. [S.l.]: Imperial College, 2011. Citado na página 49.

PENFIELD, W.; BOLDREY, E. Somatic motor and sensory representation in the cerebral cortex of man as studied by electrical stimulation. *Brain: a journal of neurology*, v. 60, n. 4, p. 389–443, 1937. Citado 3 vezes nas páginas 23, 24 e 30.

PERGE, J. A. et al. Intra-day signal instabilities affect decoding performance in an intracortical neural interface system. *Journal of Neural Engineering*, v. 10, n. 3, p. 036004, 2013. Citado na página 31.

POWLEDGE, T. M. Unlocking the secrets of the brain. *BioScience*, v. 47, n. 6, p. 330–334, 1997. Citado na página 24.

Psychology Career Center. *What is neuroscience?* 2014. Disponível em: <https://www.psychologycareercenter.org/what-is-neuroscience.html>. Acesso em: 26 de janeiro de 2019. Citado na página 23.

RAO, R. P. N. *Brain-Computer Interfacing: An introduction*. ISBN 9781139032803: Cambridge University Press, 2013. Citado 8 vezes nas páginas 29, 36, 37, 38, 39, 40, 41 e 42.

SANTOS, C. J. *Avaliação do uso de classificadores para verificação de atendimento a critérios de seleção em programas sociais*. Dissertação (Mestrado) — Universidade Federal de Juiz de Fora, Departamento de Engenharia, Modelagem Computacional. Juiz de Fora, Março 2017. Citado 2 vezes nas páginas 49 e 50.

SHARMILA, A. Signal processing algorithm for brain computer interface - a review. *World Applied Sciences*, v. 33, n. 5, p. 715–721, 2015. Citado na página 35.

SHIH, J.; KRUSIESKI, D.; WOLPAW, J. R. Brain–computer interfaces in medicine. *Mayo Clinic Proceedings*, v. 87, n. 3, p. 268–279, 2012. Citado na página 25.

SILVA, S. G. A gênese cerebral da imagem corporal: algumas considerações sobre o fenômeno dos membros fantasmas em ramachandran. *Physis: Revista de Saúde Coletiva*, v. 23, n. 1, p. 167–195, 2013. Citado na página 30.

SIULY, S.; LI, Y.; ZHANG, Y. *EEG Signal Analysis and Classification Techniques and Applications*. ISBN 9783319476520: Springer International Publishing, 2016. Citado 7 vezes nas páginas 13, 32, 33, 35, 36, 40 e 42.

SOMERSET, V. S. et al. *Intelligent and Biosensors*. ISBN 9789537619589: Inteh, 2010. Citado na página 35.

SZACHEWICZ, P. *Classification of Motor Imagery for Brain-Computer Interfaces*. Dissertação (Mestrado) — Poznan University of Technology, Faculty of Computing and Information Science. Poznań, Polônia, 2013. Citado na página 36.

Texas Instruments Incorporated. *AM335x and AMIC110 Sitara Processors: Technical Reference Manual*. 2017. Disponível em: <http://www.ti.com/lit/ug/spruh73p/spruh73p.pdf>. Acesso em: 31 de maio de 2019. Citado na página 45.

TOGA, A. W.; MAZZIOTTA, J. C. *Brain Mapping: The Systems*. ISBN 9780126925456: Academic Press, 2000. Citado na página 24.

TURI, F.; GAYRAUD, N. T.; CLERC, M. Auto-calibration of c-vep bci by word prediction. Working paper or preprint. 2020. Disponível em: <https://hal.archives-ouvertes.fr/hal-02844024>. Citado na página 52.

WOLPAW, J. R. et al. Brain–computer interfaces for communication and control. *Clinical Neurophysiology*, v. 113, n. 6, p. 767–791, 2002. Citado na página 25.

WOLPAW, J. R.; WOLPAW, E. W. *Brain-Computer Interfaces Principles and Practice*. ISBN 9780195388855: Oxford University Press, 2012. Citado 9 vezes nas páginas 13, 28, 29, 32, 34, 35, 36, 37 e 39.

ZHANG, W. et al. A review of eeg-based brain-computer interface systems design. *Brain Science Advances*, v. 4, n. 2, p. 156–167, 2018. Citado 3 vezes nas páginas 35, 36 e 37.

Apêndices

APÊNDICE A – Configuração do sistema operacional

Para executar os códigos dos algoritmos na PocketBeagel é necessário primeiramente configurar Sistema Operacional da placa. Para isso, acessou-se o site oficial da BeagleBoard para baixa a imagem mais atual do Sistema Operacional que não possui interface gráfica, pois para essa aplicação não há necessidade.

Utilizou-se o software livre BalenaEtcher para gravar a imagem no cartão de memória e, após finalizada a gravação, colocou-se o cartão na placa e conectou-a por cabo USB ao computador. Ao conectar dessa maneira, um adaptador de rede aparece no computador e a PocketBeagle começa a executar um servidor DHCP (*Dynamic Host Configuration Protocol*).

Esse servidor DHCP fornecerá a PocketBeagle um endereço IP de 192.168.7.2 ou 192.168.6.2, dependendo do tipo de sistema operacional utilizado no computador, e assim pode-se conectar de forma remota com a placa para transferir os arquivos e executá-los.

Antes de executar o programa é necessário preparar o ambiente do Sistema Operacional ao instalar os pacotes que serão utilizados. Como a imagem instalada no cartão de memória é básica, é necessário conectar a placa a internet e atualizar a lista das versões dos pacotes disponíveis.

O PocketBeagle não possui uma porta ethernet embutida ou uma rede sem fio para uso porque é uma plataforma de baixo custo. No entanto, existem várias opções para conectar o PocketBeagle à internet descritas no FAQ da página do PocketBeagle no GitHub (<https://github.com/beagleboard/pocketbeagle/wiki/FAQ#How_do_I_get_connected_to_the_Internet>).

Porém, compartilhar uma conexão de internet com a PocketBeagle não é tão simples como aparenta ser. A alternativa de compartilhar a ethernet do computador host, que é a forma mais difundida de utilização, não funcionou, mesmo utilizando vários sites diferentes como referência.

Outra forma buscada para conectar a PocketBeagle à internet foi tentando utilizar um adaptador de rede wifi, porém, para conectá-lo, era necessário adicionar uma porta USB à PocketBeagle para comunicação com o adaptador.

A PocketBeagle tem função de host USB em seu sistema operacional, mas não possui o conector físico. Dessa forma, optou-se por utilizar o módulo *USB 2.0 HUB Cape for PocketBeagle* desenvolvido pelo microwavemont by Tindie (2020).

Esta placa pode estender a capacidade da PocketBeagle ao adicionar conexão USB de 2 ou 4 portas. A função de hub é feita pelo Terminus Fe1.1s, um controlador de hub de 4 portas USB 2.0 de alta velocidade **??**.

Além disso, possui um display OLED de padrão comercial SSD1306 de 64x32 pixels que é conectado ao barramento I2C-1 de PocketBeagle. A versão utilizada nesse trabalho foi a com 4 portas USB como mostra a Figura 15.

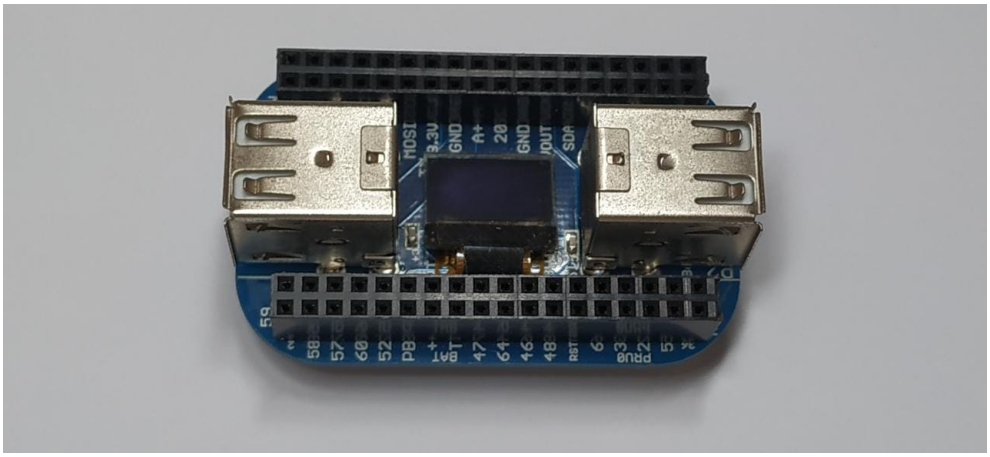


Figura 15 – Módulo *USB 2.0 HUB Cape for PocketBeagle*

Conectou-se o módulo *USB 2.0 HUB Cape* na PocketBeagle, o adaptador wifi ao módulo e ligou-se a PocketBeagle por um cabo USB ao computador. Porém, o sistema operacional da plataforma não reconheceu o adaptador wifi e, com isso, não conseguiu-se conectar a placa à internet.

Tentou-se então utilizar um adaptador de rede ethernet, como mostra a Figura 16, com o módulo *USB 2.0 HUB Cape* para verificar se dessa forma seria possível conectar a plataforma à internet para atualizar o sistema e instalar as bibliotecas necessárias para a execução dos códigos desenvolvidos.

Figura 16 – Adaptador de rede utilizado para conexão com a internet

Ao conectar o adaptador de rede ethernet ao módulo *USB 2.0 HUB Cape* e ligar a PocketBeagle por um cabo USB ao computador, verificou-se se a plataforma tinha acesso à internet ao utilizar o comando *ping* de sistemas operacionais do tipo Unix que envia pacotes de dados aos hosts da rede.

A utilização desse comando é uma maneira simples de testar, no nível mais básico, se outro sistema pode ser acessado através de uma rede e, em caso afirmativo, quanto tempo leva para que esses dados sejam trocados, confirmando assim, a conexão da PocketBeagle com a internet.

Ao verificar que a plataforma enviava e recebia pacotes pela rede, atualizou-se a lista das versões dos pacotes disponíveis no sistema operacional e instalou-se o pacote *pip*, que é o instalador de pacotes para Python.

Após instalado o *pip*, utilizou-se essa ferramenta para instalar os pacotes *numpy* e *scipy* deixando o sistema operacional pronto para executar os programas desenvolvidos para esse trabalho.

APÊNDICE B – Otimização do código em relação a memória e tempo de execução

Com o sistema atualizado e configurado, executou-se o código na PocketBeagle para verificar a quantidade de memória utilizada. Porém, passado algum tempo do inicio da execução, o programa exibiu erro de memória. Foi necessário executar o código em um computador para verificar quanto de memória o código estava ocupando ao ser executado.

Para a verificação da quantidade de memória que o programa estava ocupando, utilizou-se a ferramenta *Memory Profiler*, que é um módulo python para monitorar o consumo de memória de um processo, bem como análise linha por linha do consumo de memória para programas em python.

Porém, é importante reconhecer que o próprio criador de perfil de memória consome uma quantidade significativa de memória. Então é indicado utilizá-lo apenas na fase de desenvolvimento de programas, como foi feito, e evitá-lo na fase de produção.

Além de verificar o consumo de memória do código, também é necessário fazer um perfil do seu tempo de execução. As ferramentas mais utilizadas de criação de perfil de tempo de execução apenas calculam o tempo das chamadas das funções.

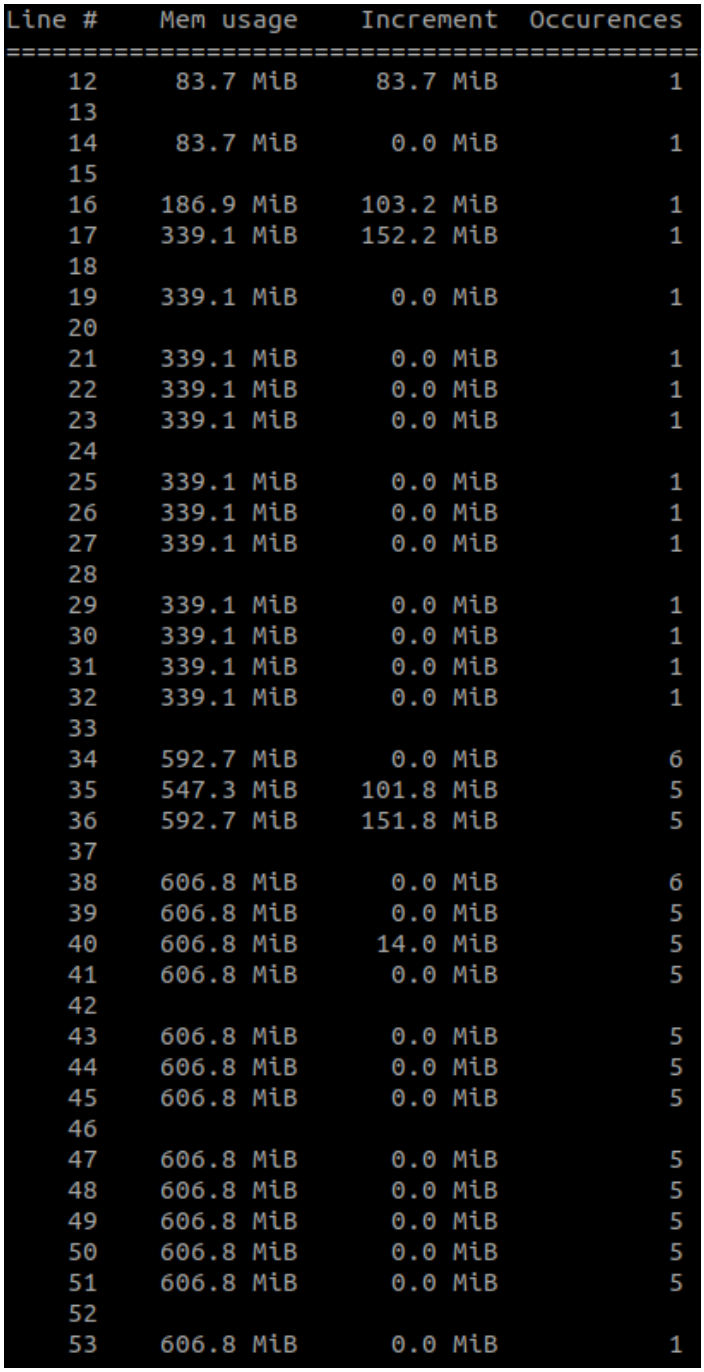
O módulo *cProfile*, por exemplo, apenas mede as chamadas de função explícitas, não métodos especiais chamados por causa da sintaxe. Consequentemente, uma operação relativamente lenta da biblioteca do *numpy* quando executada em grandes matrizes é um ponto de acesso que nunca será interrompido pelo *cProfile* porque não há chamada de função explícita nessa instrução.

Dessa forma, optou-se pelo uso do módulo *LineProfiler* que calcula o tempo de execução de cada linha individual.

Após a execução do código com o *Memory Profiler*, foi possível perceber, como mostra a Figura 17, que a partir da linha 34 do código principal a quantidade de memória utilizada passava a quantidade disponível da PocketBeagel de 512MB.

A coluna ***Line #*** mostra o número da linha analisada no arquivo e a coluna ***Mem usage*** a quantidade de memória acumulada utilizada até aquela linha. A coluna ***Increment*** exibe quanto de memória passou-se a usar a partir daquela linha e a coluna ***Occurences*** o número de vezes que essa linha foi executada.

Analisou-se também o tempo de execução do código principal, como mostra a Figura 18, e pôde-se perceber que o a linha 36 e 35 do código representam, respectivamente, o 2º e 4º maior tempo de execução.



Line #	Mem usage	Increment	Occurences
12	83.7 MiB	83.7 MiB	1
13			
14	83.7 MiB	0.0 MiB	1
15			
16	186.9 MiB	103.2 MiB	1
17	339.1 MiB	152.2 MiB	1
18			
19	339.1 MiB	0.0 MiB	1
20			
21	339.1 MiB	0.0 MiB	1
22	339.1 MiB	0.0 MiB	1
23	339.1 MiB	0.0 MiB	1
24			
25	339.1 MiB	0.0 MiB	1
26	339.1 MiB	0.0 MiB	1
27	339.1 MiB	0.0 MiB	1
28			
29	339.1 MiB	0.0 MiB	1
30	339.1 MiB	0.0 MiB	1
31	339.1 MiB	0.0 MiB	1
32	339.1 MiB	0.0 MiB	1
33			
34	592.7 MiB	0.0 MiB	6
35	547.3 MiB	101.8 MiB	5
36	592.7 MiB	151.8 MiB	5
37			
38	606.8 MiB	0.0 MiB	6
39	606.8 MiB	0.0 MiB	5
40	606.8 MiB	14.0 MiB	5
41	606.8 MiB	0.0 MiB	5
42			
43	606.8 MiB	0.0 MiB	5
44	606.8 MiB	0.0 MiB	5
45	606.8 MiB	0.0 MiB	5
46			
47	606.8 MiB	0.0 MiB	5
48	606.8 MiB	0.0 MiB	5
49	606.8 MiB	0.0 MiB	5
50	606.8 MiB	0.0 MiB	5
51	606.8 MiB	0.0 MiB	5
52			
53	606.8 MiB	0.0 MiB	1

Figura 17 – Teste inicial de consumo de memória

A coluna ***Line #*** mostra o número da linha analisada e a coluna ***Hits*** o número de vezes que essa linha foi executada. A coluna ***Time*** exibe o tempo total gasto na execução da linha nas unidades do temporizador. Nas informações do cabeçalho antes das tabelas, você verá uma linha ***Timer unit*** que fornece o fator de conversão em segundos.

A coluna ***Per Hit*** exibe a quantidade média de tempo gasto executando a linha uma vez nas unidades do temporizador e a coluna ***% Time*** a porcentagem de tempo gasto nessa linha em relação ao tempo total registrado gasto na função.

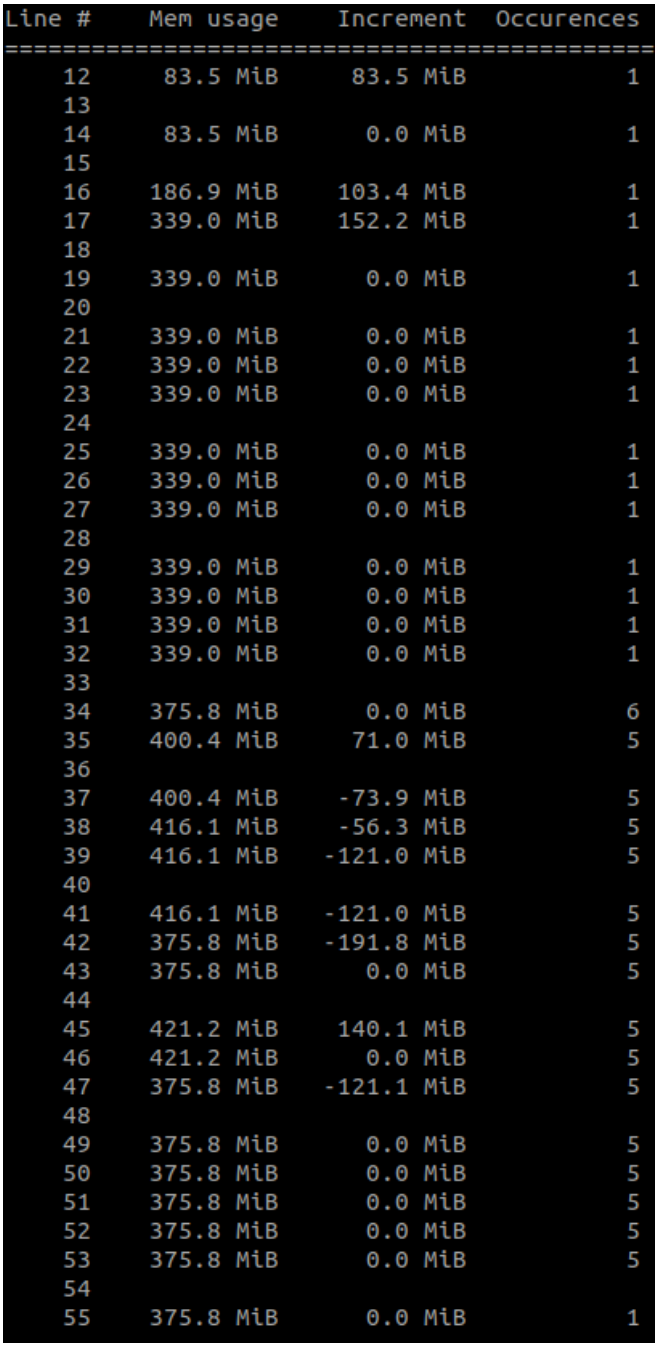
```
Timer unit: 1e-06 s

Total time: 8.84896 s
File: time_bci.py
Function: main at line 12

Line #      Hits         Time  Per Hit   % Time
================================================
    12
    13
    14         1         22.0     22.0      0.0
    15
    16         1     376186.0 376186.0      4.3
    17         1     557104.0 557104.0      6.3
    18
    19         1         22.0     22.0      0.0
    20
    21         1          1.0      1.0      0.0
    22         1          2.0      2.0      0.0
    23         1          1.0      1.0      0.0
    24
    25         1          1.0      1.0      0.0
    26         1          1.0      1.0      0.0
    27         1          1.0      1.0      0.0
    28
    29         1          2.0      2.0      0.0
    30         1          0.0      0.0      0.0
    31         1          1.0      1.0      0.0
    32         1          1.0      1.0      0.0
    33
    34         6         15.0      2.5      0.0
    35         5     684771.0 136954.2      7.7
    36         5    1014968.0 202993.6     11.5
    37
    38         6          7.0      1.2      0.0
    39         5         16.0      3.2      0.0
    40         5    5244195.0 1048839.0     59.3
    41         5         51.0     10.2      0.0
    42
    43         5     465703.0  93140.6      5.3
    44         5      21108.0   4221.6      0.2
    45         5     479230.0  95846.0      5.4
    46
    47         5       5279.0   1055.8      0.1
    48         5          9.0      1.8      0.0
    49         5        159.0     31.8      0.0
    50         5         58.0     11.6      0.0
    51         5         46.0      9.2      0.0
    52
    53         1          4.0      4.0      0.0
```

Figura 18 – Teste inicial de tempo de execução

Ao analisar o que acontecia no código na linha 34, percebeu-se que os datasets dos 5 sujeitos a serem analisados estavam sendo armazenados em variáveis ao mesmo tempo, porém eram usados em momentos diferentes. Com isso, usava-se uma quantidade de memória desnecessária que não era utilizada assim que armazenada.

Dessa forma, otimizou-se o código principal para armazenar apenas o EEG filtrado do sujeito que seria analisado no momento, liberando o espaço ao término do uso para armazenar o EEG filtrado do próximo sujeito. Com essas modificações, consegui-se o perfil

de memória exibido na Figura 19.



Line #	Mem usage	Increment	Occurences
12	83.5 MiB	83.5 MiB	1
13			
14	83.5 MiB	0.0 MiB	1
15			
16	186.9 MiB	103.4 MiB	1
17	339.0 MiB	152.2 MiB	1
18			
19	339.0 MiB	0.0 MiB	1
20			
21	339.0 MiB	0.0 MiB	1
22	339.0 MiB	0.0 MiB	1
23	339.0 MiB	0.0 MiB	1
24			
25	339.0 MiB	0.0 MiB	1
26	339.0 MiB	0.0 MiB	1
27	339.0 MiB	0.0 MiB	1
28			
29	339.0 MiB	0.0 MiB	1
30	339.0 MiB	0.0 MiB	1
31	339.0 MiB	0.0 MiB	1
32	339.0 MiB	0.0 MiB	1
33			
34	375.8 MiB	0.0 MiB	6
35	400.4 MiB	71.0 MiB	5
36			
37	400.4 MiB	-73.9 MiB	5
38	416.1 MiB	-56.3 MiB	5
39	416.1 MiB	-121.0 MiB	5
40			
41	416.1 MiB	-121.0 MiB	5
42	375.8 MiB	-191.8 MiB	5
43	375.8 MiB	0.0 MiB	5
44			
45	421.2 MiB	140.1 MiB	5
46	421.2 MiB	0.0 MiB	5
47	375.8 MiB	-121.1 MiB	5
48			
49	375.8 MiB	0.0 MiB	5
50	375.8 MiB	0.0 MiB	5
51	375.8 MiB	0.0 MiB	5
52	375.8 MiB	0.0 MiB	5
53	375.8 MiB	0.0 MiB	5
54			
55	375.8 MiB	0.0 MiB	1

Figura 19 – 2º teste de consumo de memória

Como pode-se perceber, o máximo de memória utilizada para treinar e testar os EEGs dos pacientes foi de 421.2MB, o que fica dentro do limite de memória disponível na PocketBeagle. Também gerou-se o perfil de tempo de execução do novo código para verificar se houveram grandes alterações com as modificações realizadas no código.

Como pode-se observar pela Figura 20, houve um pequeno aumento no tempo de execução de 0,32115s e as proporções dos tempos de execução de cada linha também sofreram leves alterações. Entretanto, pode-se perceber que as modificações realizadas não

alteraram de forma expressiva o desempenho do programa.

```
Timer unit: 1e-06 s

Total time: 9.17011 s
File: time_bci.py
Function: main at line 12

Line #      Hits         Time  Per Hit   % Time
================================================
    12
    13
    14         1         21.0     21.0      0.0
    15
    16         1     378369.0 378369.0      4.1
    17         1     556898.0 556898.0      6.1
    18
    19         1         25.0     25.0      0.0
    20
    21         1          2.0      2.0      0.0
    22         1          2.0      2.0      0.0
    23         1          1.0      1.0      0.0
    24
    25         1          1.0      1.0      0.0
    26         1          1.0      1.0      0.0
    27         1          1.0      1.0      0.0
    28
    29         1          2.0      2.0      0.0
    30         1          1.0      1.0      0.0
    31         1          1.0      1.0      0.0
    32         1          1.0      1.0      0.0
    33
    34         6         10.0      1.7      0.0
    35         5     754795.0 150959.0      8.2
    36         5         22.0      4.4      0.0
    37         5    5455397.0 1091079.4     59.5
    38         5         46.0      9.2      0.0
    39         5     474031.0  94806.2      5.2
    40         5       3491.0    698.2      0.0
    41
    42         5    1127464.0 225492.8     12.3
    43         5     393400.0  78680.0      4.3
    44         5       5687.0   1137.4      0.1
    45
    46         5      13809.0   2761.8      0.2
    47         5       6347.0   1269.4      0.1
    48         5          8.0      1.6      0.0
    49         5        142.0     28.4      0.0
    50         5         69.0     13.8      0.0
    51         5         56.0     11.2      0.0
    52
    53         1          7.0      7.0      0.0
```

Figura 20 – 2º teste de tempo de execução