MEDUSA: UNA NUEVA HERRAMIENTA PARA EL DESARROLLO DE SISTEMAS BRAIN-COMPUTER INTERFACE BASADA EN PYTHON

Eduardo Santamaría-Vázquez, Víctor Martínez-Cagigal, Roberto Hornero
Grupo de Ingeniería Biomédica, E.T.S.I. de Telecomunicación, Universidad de Valladolid, Paseo Belén 15, 47011, Valladolid, España, { *eduardo.santamaria@gib.tel.uva.es*, *victor.martinez@gib.tel.uva.es*, *robhor@tel.uva.es*}

Resumen

En este estudio se presenta MEDUSA, una nueva plataforma para la implementación de sistemas Brain-Computer Interface (BCI), utilizando como señal de control los potenciales evocados P300 generados mediante el paradigma odd-ball visual. Las principales características de MEDUSA son: i) diseño modular, conectando las distintas partes de la aplicación mediante estructuras simples; ii) arquitectura especialmente diseñada para entornos de investigación, que permite implementar nuevos paradigmas de estimulación, métodos de pre-procesado de señal, o algoritmos de extracción, selección y clasificación de características de manera sencilla e integrarlos rápidamente en el flujo de ejecución; iii) desarrollo en Python: un lenguaje multiplataforma muy utilizado en investigación por su simplicidad y versatilidad, disponiendo además de gran cantidad de librerías que disminuyen el tiempo de desarrollo; iv) amplia documentación, con ejemplos de aplicación y código detalladamente comentado; v) interfaz gráfica atractiva y moderna, que permite modificar de manera sencilla gran cantidad de parámetros de la plataforma.

Palabras Clave: Brain-Computer Interface (BCI), Electroencefalografía (EEG), potenciales evocados P300.

1 INTRODUCCIÓN

Los sistemas *Brain-Computer Interface* (BCI) permiten la comunicación en tiempo real entre una persona y un dispositivo electrónico midiendo la actividad neuronal, sin la necesidad de que intervengan músculos o nervios periféricos [1]. Aunque existen multitud de métodos para registrar la actividad cerebral, en la práctica normalmente se emplea el electroencefalograma (EEG) debido a que se adquiere con un equipo portable, no invasivo y de bajo coste en comparación con otras técnicas disponibles [1].

En la actualidad, la principal motivación de los sistemas BCI es el aumento de la calidad de vida y la independencia de personas con discapacidad física severa, dotándolas de un nuevo canal de comunicación con el entorno [1]. Especialmente importantes en este campo son las aplicaciones que utilizan como señal de control los potenciales evocados P300, generados mediante el paradigma *odd-ball* visual [1]. Este paradigma permite discriminar las intenciones del usuario entre un gran número de comandos posibles, permitiendo que personas con grave discapacidad física puedan utilizar prótesis, sillas de ruedas, sistemas domóticos, ordenadores, etc.

La arquitectura típica de un sistema BCI generalmente consta de tres elementos: i) *hardware* para el registro y la transmisión del EEG; ii) plataforma *software* encargada de estimular al usuario, procesar, y analizar la señal de EEG para identificar sus intenciones y convertirlas en un comando de la aplicación; y iii) aplicación a controlar, que debe recibir el comando de salida de la plataforma y ejecutarlo. La calidad de estos tres elementos básicos, así como la habilidad personal del usuario, determinará el rendimiento del sistema BCI implementado y su utilidad en la vida real [2].

Una de las mayores limitaciones a la hora de realizar estudios de investigación en BCI es la plataforma utilizada, que debe cumplir unos requisitos exigentes: visualizar la señal EEG recibida; implementar métodos de extracción, selección y clasificación de características en tiempo real; y estimular visualmente y realimentar al usuario de forma sincronizada con los procesos anteriores. Estos requisitos hacen que el desarrollo de la plataforma suponga una gran inversión de horas de trabajo y sea de gran complejidad técnica. Además, los métodos de procesado de señal son una de las principales líneas de investigación en BCI, por lo que es esencial disponer de la posibilidad de realizar pruebas de distintos métodos rápidamente, sin tener que alterar otras partes de la aplicación. Por esta razón se revela fundamental seguir los principios de simplicidad y modularidad, es decir, que la plataforma se

componga de elementos autónomos conectados por estructuras coherentes y simples que permitan actualizar distintas partes de la aplicación, dejando las otras inalteradas [3].

Con el objetivo de resolver estas limitaciones, el Grupo de Ingeniería Biomédica (GIB) de la Universidad de Valladolid está desarrollando MEDUSA: una plataforma *software*, desarrollada en Python, para la implementación de aplicaciones BCI que utilizan como señal de control los potenciales P300, generados mediante un paradigma *odd*-ball visual. El diseño de MEDUSA es simple y modular, lo que minimiza el acoplamiento entre las distintas partes de la aplicación. Esto permite introducir nuevos módulos y realizar cambios en el código de la manera más sencilla posible, lo que hace de MEDUSA una herramienta versátil y escalable, algo especialmente valioso en entornos de investigación.

2 PLATAFORMAS BCI

Actualmente, existen plataformas disponibles para la implementación de sistemas BCI. En concreto, se han identificado ocho proyectos de este tipo: BCI2000, OpenViBE, TOBI Common Implementation Platform (CIP), BCILAB, BCI++, xBCI, BF++ y las librerias Mushu, Wyrm y Pyff [4], [5].

BCI2000 es una plataforma de propósito general para el diseño de experimentos BCI [2]. Este software no es libre, pero su código fuente y sus ejecutables compilados están exentos de pago para aplicaciones con propósitos educacionales y de investigación. Contiene todas las funciones para el diseño de un sistema BCI completo, y se compone de cuatro módulos: i) *Source Module*, que se encarga de obtener la señal de EEG; ii) *Signal Processing Module*, que abarca el pre-procesado, la extracción y la clasificación de características; iii) *User Application Module*, que permite la interacción del usuario con la aplicación; y iv) *Operator Module*, que facilita la visualización de datos y la configuración del sistema. Ha sido desarrollada en C++.

OpenViBE también contiene todos los elementos necesarios para implementar un sistema BCI completo [6]. Además, dispone de una interfaz gráfica que permite el diseño de un sistema BCI de manera visual, donde las distintas etapas del procesado de señal se representan como cajas que se pueden conectar unas con otras. Al igual que BCI2000, ha sido desarrollada en C++.

BCI2000 y OpenViBE son las plataformas más utilizadas actualmente por contar con más funciones que el resto, extensa documentación y una amplia comunidad que da soporte a las aplicaciones. Por su parte, BCI++, xBCI y BF++ también permiten la implementación de un sistema BCI completo, aunque con menos opciones que las dos anteriores y una documentación más reducida, por lo que cuentan con una comunidad mucho más pequeña.

TOBI y BCILAB son plataformas más específicas y se utilizan como complemento a otros *softwares*. TOBI es una interfaz que facilita la conexión de las distintas partes de un sistema BCI distribuido a través de la red, disminuyendo el tiempo de desarrollo y estandarizando los tipos de datos [4]. BCILAB es un conjunto de herramientas basadas en MATLAB que permiten desarrollar y probar nuevos métodos de procesado o extracción y clasificación de características rápidamente. Sin embargo, no incluye módulos de adquisición de la señal o paradigmas de estimulación, por lo que debe complementarse con otras plataformas compatibles para crear un sistema BCI funcional [7].

Por último, también existe un conjunto de tres librerías escritas en Python, que tienen el objetivo de facilitar la implementación de un sistema BCI: i) Mushu, que contiene las funciones necesarias para la adquisición de la señal de EEG con varios amplificadores [8]; ii) Wyrm, que contiene los módulos necesarios para procesar la señal [5]; y iii) Pyff, que permite la estimulación del usuario y la realimentación necesaria para completar las funciones de un sistema BCI [9]. Estas librerías forman, conjuntamente, una plataforma BCI completa. No obstante, están escritas en Python 2.7, una versión no compatible con la actual (Python 3.6), y su última actualización se produjo hace varios años, por lo que es proyecto sin soporte actual.

3 ARQUITECTURA DE MEDUSA

3.1 CARACTERÍSTICAS PRINCIPALES

MEDUSA es una plataforma BCI que tiene como propósito proporcionar una alternativa para el desarrollo de aplicaciones BCI. Las principales características de esta plataforma se detallan a continuación.

3.1.1 Estructura genérica

MEDUSA implementa las principales etapas de una plataforma BCI: adquisición y pre-procesado de la señal, extracción, selección y clasificación de características, y paradigma de estimulación. Las funcionalidades específicas implementadas en cada uno de estos módulos se explican en el apartado 3.2.

3.1.2 Modularidad

Una de las principales características de MEDUSA es que ha sido diseñada siguiendo una arquitectura modular, compuesta de elementos autónomos conectados por estructuras simples, que permitan realizar cambios e implementar nuevos métodos sin alterar el resto de la aplicación.

3.1.3 Diseño específico para investigación

MEDUSA esta específicamente diseñada para facilitar la labor investigadora. Su arquitectura permite probar nuevos métodos *online* de pre-procesado, extracción de características y clasificación de manera rápida gracias al bajo grado de acoplamiento entre las distintas partes de la aplicación. Además, se ha puesto especial énfasis en comentar detalladamente el código y en generar una documentación extensa, con ejemplos que ilustran el funcionamiento de la plataforma.

3.1.4 Desarrollada en Python

MEDUSA es una plataforma complemente desarrollada en Python 3, característica que hace que sea portable a cualquier sistema operativo o plataforma *hardware* compatible. Actualmente, este lenguaje interpretado de alto nivel es uno de los más usados en investigación, debido su simplicidad y su filosofía *open-source*. Además, cuenta con una amplia comunidad que desarrolla un gran número de herramientas y librerías específicas que facilitan el análisis y el procesado de la señal, como SciPy o Numpy. También permite crear interfaces gráficas mediante PyQt5.

3.2 ESTRUCTURA

En esta sección se describen los módulos que componen MEDUSA y los métodos y funciones implementados en cada uno de ellos.

3.2.1 Módulo de adquisición de la señal

El módulo de adquisición de señal implementa las funciones necesarias para recibir la señal de EEG desde el amplificador y retransmitirla al resto de la plataforma. MEDUSA es compatible con cualquier amplificador utilizando *Lab Streaming Layer* (LSL), un protocolo open-source para la adquisición de datos en tiempo real. Este *software*, desarrollado por el *Swartz Center for Computational Neuroscience* de la Universidad de California San Diego (UCSD), añade una capa de abstracción sobre el protocolo de comunicación específico de cada amplificador, permitiendo homogeneizar la comunicación con la aplicación. Actualmente, el repositorio de LSL cuenta con una amplia biblioteca de aplicaciones que crean esta capa de abstracción para multitud de dispositivos (por ej. g.USBamp, Emotiv, Enobio, Biosemi, etc.), haciendo que MEDUSA sea compatible con todos ellos, a cualquier frecuencia de muestreo, y con cualquier configuración de canales. Sin embargo, por el momento, sólo se ha comprobado su funcionamiento con el amplificador g.USBamp de la empresa g.Tec (Guger Technologies, Austria).

3.2.2 Módulo de pre-procesado

El módulo de pre-procesado implementa métodos para: i) mejorar la calidad de la señal de EEG, eliminando ruido, artefactos y componentes de continua de cada canal; y ii) seleccionar determinadas bandas en frecuencia o épocas de señal que sean de especial interés. Actualmente, se incluyen dos clases de métodos: i) filtros espaciales, en concreto se implementa el filtro *Common Average Reference* (CAR); y ii) filtros frecuenciales paso banda tipo Butterworth, y un filtro *notch* para rechazar ciertas bandas estrechas (por ej. 50 Hz).

3.2.3 Módulo de extracción de características

El módulo de extracción de características contiene métodos que extraen información de la señal. En concreto, se ha implementado un método de sub-muestreo. Una configuración típica sería usar un sub-muestreo a 20 Hz, épocas de 1 segundo registradas con una frecuencia de muestreo de 256 Hz, y una configuración de 8 canales. Con esta configuración, se extraerían 160 características por cada época de señal.

3.2.4 Módulo de selección de características

El módulo de selección contiene algoritmos que seleccionan las características más significativas de las extraídas anteriormente como entrada al clasificador. Actualmente cuenta con tres métodos tipo *wrapper*: i) Forward-Selection (FS), que incluye las características si cumplen un determinado criterio de entrada; ii) Backward-elimination (BE), que elimina las características si cumplen un determinado criterio de salida; y iii) Stepwise-Forward-Selection-Backward-Elimination (Stepwise FS-BE), que combina los dos métodos anteriores, incluyendo en cada iteración una característica con FS para posteriormente evaluar el conjunto seleccionado con BE [10].

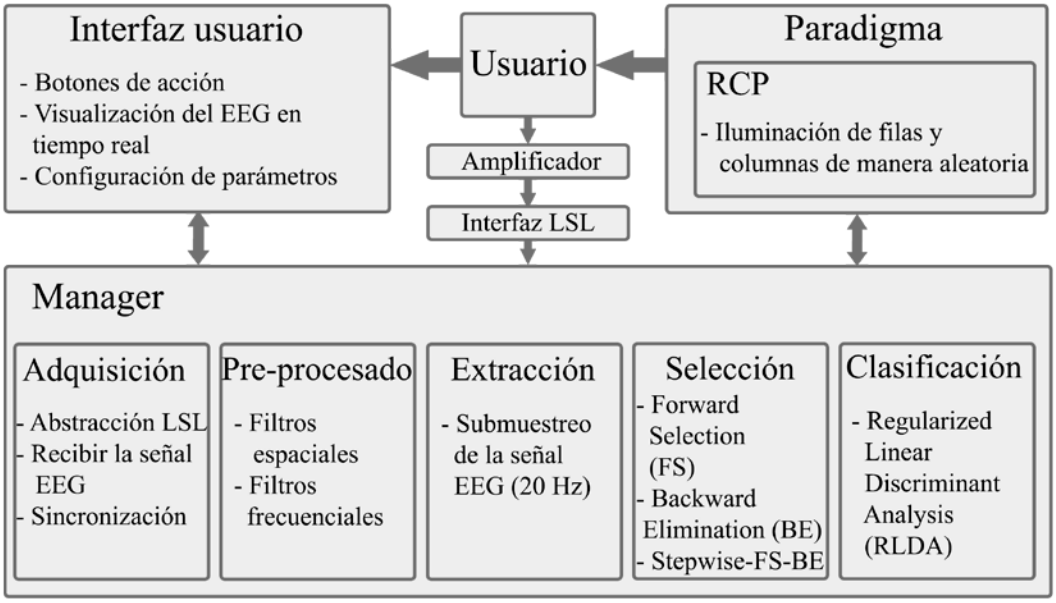

Figura 1: Módulos utilizados por cada proceso de MEDUSA, resumiendo sus funciones.

3.2.5 Módulo de clasificación

En el módulo de clasificación se encuentran los algoritmos que se utilizan para determinar si en una determinada época de señal existe un P300 o no, en base a las características seleccionadas por el módulo de selección de características. Por el momento, el único clasificador implementado es el *Regularized Linear Discriminant Analysis* (RLDA). Este clasificador lineal calcula una proyección sobre un espacio vectorial que: i) reduce la dimensionalidad de los datos; ii) maximiza la covarianza entre clases; y iii) minimiza la covarianza dentro de la clase [11].

3.2.6 Módulo de paradigmas

Este módulo contiene los paradigmas *odd-ball* utilizados para estimular al usuario y generar los potenciales evocados P300. Por el momento, se ha implementado el paradigma *Row Column Paradigm* (RCP) [12], que ilumina de manera aleatoria las filas y columnas de una matriz de comandos. Además, el paradigma debe determinar los tiempos en los que comienza cada estímulo, así como la fila o columna en la que ocurre.

3.3 DETALLES DE IMPLEMENTACIÓN

La actividad de MEDUSA se desarrolla en tres procesos sincronizados que realizan todas las operaciones necesarias para el funcionamiento de la plataforma. En la Figura 1 se ilustran las funciones de estos tres procesos. En el proceso principal se ejecuta la interfaz de usuario, desarrollada con PyQT5. Esta interfaz ofrece la posibilidad al usuario de interactuar con la aplicación y configurar sus parámetros. También se encarga de la visualización en tiempo real de la señal de EEG, y de informar al usuario del estado de MEDUSA en cada momento. Las características y posibilidades de esta interfaz se detallan en el apartado 4. En un segundo proceso, implementado en una clase llamada *Manager*, se recibe la señal de EEG y los datos de estimulación del paradigma RCP para posteriormente aplicar los métodos de pre-procesado, extracción, selección y clasificación de características que permiten identificar las intenciones del usuario en cada *trial*. Por último, en el tercer proceso se ejecuta la interfaz gráfica del paradigma, y se guardan los tiempos en los que se ha iluminado cada fila o columna para su posterior envío al proceso M*anager*. La sincronización entre los tres procesos se realiza mediante tres variables de estado, compartidas por todos los procesos: i) variable de estado del amplificador; ii) variable de estado del paradigma; y iii) variable de estado de la ejecución. El usuario puede cambiar el valor de estas variables para manejar el flujo de trabajo de la plataforma desde la interfaz gráfica.

4 RESULTADOS Y DISCUSIÓN

El resultado del desarrollo de MEDUSA se muestra en las Figuras 2 y 3. En la Figura 2 se muestra el panel principal de MEDUSA. Los botones de acción permiten al usuario interactuar con la aplicación, conectar y desconectar el amplificador, iniciar el paradigma, o empezar, pausar y parar una ejecución. La ventana de parámetros principales permite configurar la siguiente ejecución. El panel de información (*Log window*) indica el estado de la aplicación al usuario, mostrando la información

Figura 2: Panel principal de MEDUSA. (a) Botones de acción. (b) Panel de configuración principal. (c) Panel de información. (d) Representación del EEG en tiempo real.

Figura 3: Interfaz gráfica del paradigma RCP implementado en MEDUSA.

más relevante en cada momento. Por último, se muestra la gráfica que representa el EEG en tiempo real, que incluye opciones como la posibilidad de auto-escalado, elección los canales mostrados, etc. En la Figura 3 se muestra la interfaz gráfica del paradigma. Se trata de un RCP clásico, que puede funcionar en modo entrenamiento, o *copy-mode*, donde el usuario debe seleccionar una secuencia pre-establecida de comandos y permite optimizar el clasificador, o modo *online*, donde las selecciones son elegidas libremente por el usuario. Se incluye también la posibilidad de hacer registros basales.

En la tabla 1 se muestra una comparación entre las principales plataformas BCI analizadas en el punto 2 y MEDUSA. Entre las opciones disponibles destacan BCI2000 y OpenViBE por implementar todas las etapas de un sistema BCI y tener soporte actual. Además, son plataformas muy completas, que permiten utilizar varias señales de control e implementar aplicaciones muy diferentes entre sí. Sin embargo, esta característica también hace que sean plataformas complejas, y durante su utilización en proyectos de investigación se han identificado algunas limitaciones que han motivado el desarrollo de MEDUSA. En concreto: i) código muy extenso, poco ordenado y no detalladamente comentado; ii) alto acoplamiento entre funciones de un mismo módulo, impidiendo realizar cambios y pruebas de manera sencilla; iii) desarrolladas en C++, un lenguaje muy eficiente, pero de bajo nivel y con una sintaxis compleja que aumenta el tiempo de desarrollo. MEDUSA resuelve estas limitaciones por: i) ser una plataforma más específica y simple; ii) implementar más módulos, con menos funciones y más concretas; y iii) haber sido desarrollada en Python: un lenguaje multiplataforma con una sintaxis sencilla, y gran cantidad de librerías que disminuyen el tiempo de desarrollo.

5 CONCLUSIONES

En esta comunicación se ha presentado MEDUSA: una nueva plataforma para la implementación de sistemas BCI que utilizan como señal de control los potenciales evocados P300, generados mediante el paradigma *odd-ball* visual. El diseño de MEDUSA es modular, conectando las distintas partes de la aplicación mediante estructuras coherentes y simples. Una de sus principales características es que está especialmente diseñada para equipos de

Tabla 1: Comparación entre las principales plataformas BCI.

Plataforma	Sistema Operativo	Requisitos	Funciones	Lenguaje	Soporte
BCI2000	Windows	-	Completas	C++	Sí
OpenViBE	Windows, Linux	-	Completas	C++	Sí
TOBI	Windows, Linux	-	Interfaz de comunicación	C++	No
BCILAB	Multiplataforma	MATLAB®	Procesado de señal	MATLAB®	Sí
Mushu	Multiplataforma	Python 2	Adquisición de señal	Python	No
Wyrm	Multiplataforma	Python 2	Procesado de señal	Python	No
Pyff	Multiplataforma	Python 2	Paradigma	Python	No
MEDUSA	**Multiplataforma**	**Python 3**	**Completas (P300)**	**Python**	**En desarrollo**

investigación, ya que su arquitectura permite implementar nuevos paradigmas de estimulación, métodos de pre-procesado de señal, y algoritmos de extracción, selección y clasificación de características de manera sencilla, e integrarlos rápidamente en el flujo de ejecución. También permite la configuración de gran cantidad de parámetros mediante una interfaz gráfica atractiva y moderna. MEDUSA es un proyecto a largo plazo, que se encuentra actualmente bajo desarrollo. En el futuro se plantea su distribución bajo licencia *open-source*, la implementación de nuevos métodos de procesado de señal, y la posibilidad de utilizar otras señales de control diferentes de los potenciales evocados P300, como los ritmos sensoriomotores mu y beta.

Agradecimientos

Este estudio ha sido financiado por los proyectos TEC2014-53196-R y DPI2017-84280-R del Ministerio of Economía y Competitividad y FEDER, el proyecto "Análisis y correlación entre el genoma completo y la actividad cerebral para la ayuda en el diagnóstico de la enfermedad de Alzheimer" (Programa Operativo de Cooperación Transfronteriza España-Portugal, POCTEP, 2014-2020) de la Comisión Europea y FEDER, y el proyecto VA037U16 de la Junta de Castilla y León y FEDER. Víctor Martínez-Cagigal es beneficiario de una ayuda PIF-UVa de la Universidad de Valladolid.

Referencias

[1] T. M. Wolpaw, J. R.; Birbaumer, N.; McFarland, D. J.; Pfurtscheller, G.; Vaughan, "Brain Computer Interfaces for communication and control," *Clin. Neurophysiol.*, vol. 4, no. 113, pp. 767–791, 2002.

[2] G. Schalk, D. J. McFarland, T. Hinterberger, N. Birbaumer, and J. R. Wolpaw, "BCI2000: A general-purpose brain-computer interface (BCI) system," *IEEE Trans. Biomed. Eng.*, vol. 51, no. 6, pp. 1034–1043, 2004.

[3] P. America, *Object-oriented software construction*, vol. 12, no. 1. 1989.

[4] A. N. Brendan Z. Allison, Stephen Dunne, Robert Leeb, José Del R.Millán, *Towards Practical Brain-Computer Interfaces: Bridging the Gap from Research to Real-World Applications*, vol. 1. 2015.

[5] B. Venthur, S. Dähne, J. Höhne, H. Heller, and B. Blankertz, "Wyrm: A Brain-Computer Interface Toolbox in Python," *Neuroinformatics*, vol. 13, no. 4, pp. 471–486, 2015.

[6] Y. Renard *et al.*, "OpenViBE : An Open-Source Software Platform to Design , Test , and Use Brain – Computer Interfaces in Real and Virtual Environments," *Presence*, vol. 19, no. 1, pp. 35–53, 2010.

[7] C. A. Kothe and S. Makeig, "BCILAB: A platform for brain-computer interface development," *J. Neural Eng.*, vol. 10, no. 5, 2013.

[8] B. Venthur and B. Blankertz, "Mushu, a free- and open source BCI signal acquisition, written in Python," 2016, pp. 1786–1788.

[9] B. Venthur *et al.*, "Pyff – A Pythonic Framework for Feedback Applications and Stimulus Presentation in Neuroscience," *Front. Neurosci.*, vol. 4, no. December, pp. 1–17, 2010.

[10] J. D. Jobson, *Applied multivariate data analysis: volume I: Regression and Experimental Design*. Springer Science & Business Media, 1999.

[11] K. Fukunaga, *Introduction to statistical pattern recognition*. Academic press, 2013.

[12] L. A. Farwell and E. Donchin, "Talking off the top of your head: toward a mental prosthesis utilizing event-related brain potentials," *Electroencephalogr. Clin. Neurophysiol.*, vol. 70, no. 6, pp. 510–523, 1988.