Position: A Novice Oriented Dual-Modality Programming Tool for Brain-Computer Interfaces Application Development

Ajay Mehul
Dept. of Computer Science
University of Alabama
Tuscaloosa, AL, US
aanbuselvam@crimson.ua.edu

Nicholas Cioli
Dept. of Computer Science
University of Alabama
Tuscaloosa, AL, US
ncioli@crimson.ua.edu

Chris S. Crawford
Dept. of Computer Science
University of Alabama
Tuscaloosa, US
crawford@cs.ua.edu

Andre Denham
College of Education
University of Alabama
Tuscaloosa, US
adenham@ua.edu

***Abstract*—Brain-Computer Interfaces (BCI) are commonly used to translate brain activity to commands. The emergence of more affordable electroencephalography (EEG) hardware is gradually making neurotechnology more accessible to a broader audience. However, there is limited work investigating ways to design software that supports novice BCI developers. To address these issues, we have designed NeuroSquare, a hybrid block-flow based programming tool that provides a live environment designed to assist educators with introducing novice programmers to BCI. NeuroSquare divides applications into three parts: signal acquisition, signal processing, and application logic. It provides a flow-based environment for signal processing and a block-based environment for the application logic. The flow-based section provides the freedom to process EEG signals. Furthermore, the hybrid design may also provide insights into the efficacy of dual-modality environments in the context of neurofeedback application development. This paper discusses NeuroSquares software design, limitations, and future work.**

***Index Terms*—Block-based programming; Programming Environments; Design; CS Education; BCI; EEG**

I. Introduction

As computing continues to spread across various disciplines, interdisciplinary educational approaches have gradually emerged. Educators have previously explored courses that attempt to combine computer science concepts with areas such as robotics [1], art [2], law [3], and biology [4]–[6]. Recently, physiological sensing hardware technologies, such as electrocardiogram (ECG, electrical heart activity), electromyography (EMG, electrical muscle activity), and electroencephalogram (EEG, electrical brain activity), have gradually become more affordable. This class of emerging consumer-grade hardware presents more affordable opportunities to users interested in learning more about novel physiological sensor technologies. Furthermore, affordable physiological sensors may supplement an interdisciplinary curriculum that involves aspects of computer science, human-computer interaction, and physiology. While cost-related barriers may be gradually fading, current software platforms for physiological application development are commonly designed for domain experts. Furthermore, previous block-based-only approaches to physiological application development may hinder users' ability to design custom signal processing pipelines [7]. In this paper, we present a dual-modality concept that explores the combination of block- and flow-based visual interfaces to assist users with designing both the application logic (e.g. game rules) and signal processing procedures of a physiological-computing application.

II. Background

A. BCI Tools

Several tools have been developed for programming BCI (brain-computer interface) applications with varying levels of difficulty. BCI2000 is a visual programming tool that allows users to develop BCI applications but requires C++ knowledge in order to do so [8]. OpenVibe provides a visual, flow-based environment for creating BCI applications [9] but requires experience to set up and interface with external software. BCILAB is a BCI development toolbox for MATLAB [10] but requires MATLAB which can be expensive for novice programmers. Moreover, these tools target experienced researchers and are not designed for novice programmers. Ease-of-setup and ease-of-use are significant design considerations when designing tools for users unfamiliar with existing physiological computing workflows.

B. Psychology of programming modalities

Green et al. and Vessey et al. developed extensive research bodies on the effects of the notational structure of programming languages and tools [11]–[14]. Green et al. summed up their work into two maxims [15]:

- Every notation highlights some kinds of information at the expense of obscuring other kinds.
- When seeking information, there must be a cognitive fit between the mental representations and the external representation.

A flow-based programming tool obscures the program structure while a block-based programming tool obscures the flow of data.

978-1-7281-4849-6/19/$31.00 ©2019 IEEE

Highlighting dataflow can be useful for data processing sections of a program, as shown by existing data-intensive programs such as LabView, Simulink, and OpenVibe.

C. Block-based educational tools

Block-based programming languages preserve structured flow in programs and are employed in several novice programming tools, like Scratch [7], [16], [17]. Weintrop et al. studied the advantages of these types of programming languages over those of traditional text-based languages for novice programmers [18] and theorize that, while novices find that there are limitations with block-based tools, those novices show higher interest and better performance in computer science over novices learning with text-based programming languages. In a similar vein, Crawford et al. developed a block-based BCI programming tool for novice programmers and suggest that while students found Neuroblock limiting due its lack of signal processing capabilities, the tool improved the confidence of novice BCI programmers [7], [19].

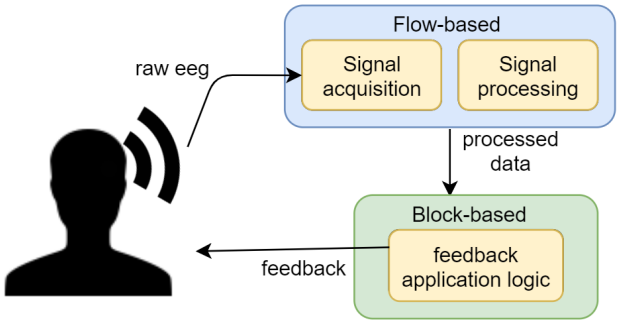
D. Dual-modality programming

Weintrop et al. researched dual-modality programming environments for learners, comparing a hybrid, block-text-based programming tool with exclusive text or block-based environments [20]. Their research showed that students perceived the hybrid environment to be more authentic. Weintrop et al. hypothesize that this could be due to how hybrid environments highlight the polymorphic nature of programming in how it exists in various modalities, environments, and technologies.

III. Design

In this section, we present an example physiological-computing development environment that leverages EEG data in order to develop BCI applications. Wolpaw proposed the one of the popular [21], [22] designs for BCI systems which separated applications into three parts: signal acquisition, signal processing, and application [23]. We have designed a dual-modality programming environment which provides a flow-based environment for data acquisition/signal processing and a block-based environment for application logic (see Fig. 1).

Fig. 1. System design for applications developed on our hybrid environment



IV. NeuroSquare

NeuroSquare is a front-end web application that provides a hybrid environment for programming BCI applications. Kelleher and Pausch [24] suggest that programming barriers can be lowered "by simplifying the mechanics of programming, by providing support for learners, and by providing students with motivation to learn to program" (p. 131). NeuroSquare's hybrid block- and flow-based approach prevents syntactical errors and provides engaging feedback by allowing users to modify the program in real time with no added set up.

A. EEG Device

Real-time EEG data drives the interactivity of this application and requires a compatible BCI device. Muse is an affordable, non-invasive EEG headset that streams data through Bluetooth. The Muse collects EEG data from four electrodes (TP9, AF7, AF8, TP10) and 1 reference electrode (FPZ) based on the international 10-20 electrode positioning system and streams it at 256 Hz over Bluetooth. Basic signal processing can be performed using JavaScript libraries in under 4ms [25] which makes the processing functions near real-time.

B. Flow-based component

NeuroSquare uses Rete.js (a JavaScript framework for visual programming) to implement its flow-based programming features. The tool provides users with drag and drop boxes that can be connected to each other. Each box in NeuroSquare has input sockets, controls, and output sockets. Controls in each box are Vue.js (a JavaScript front end framework) applications that execute the purpose of the box using its inputs and return information using the output sockets.

The boxes provided by NeuroSquare can be categorized into two types: signal acquisition and signal processing. The following subsections provide details about the functions of these boxes.

1) Signal Acquisition: Signal acquisition boxes provide the communication component for NeuroSquare to receive data from EEG headsets or the user's computer. The Muse Device box streams data directly to the browser from the Muse Headset using web-Bluetooth. The CSV Load box allows users to upload a CSV file to the browser. The output sockets make the raw EEG data available to other boxes for processing or display.

2) Signal Processing: Signal processing boxes abstract algorithms used for analyzing the raw EEG data. Signal processing boxes take input from signal acquisition boxes and other signal processing boxes to compute data for interactive feedback boxes. Bci.js [25] is an open source JavaScript library that provides algorithms for real-time analysis for EEG data. NeuroSquare provides boxes for Bci.js algorithms (see Fig. 3) and basic mathematical operations which allow users to create logic for their application. Although there is limited research on performing heavy, real-time computation in the browser, prior research has shown that browsers can handle simple EEG processing in real time [25].

Fig. 2. Example application with Flow-based environment calculating Alpha power and outputting it to a hybrid variable (alpha), Block-based environment controlling a player using hybrid variable (alpha)

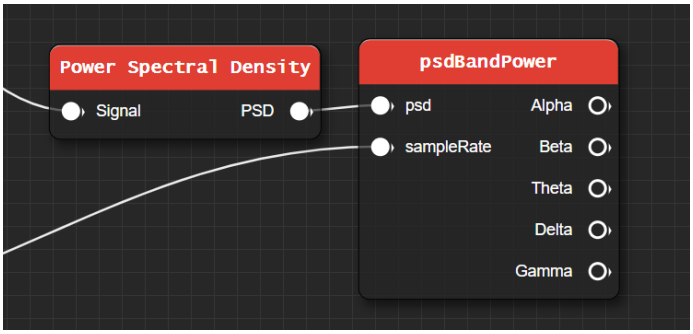


Fig. 3. Power Spectral Density box (left), Band Power Box(right)

C. Block-based component

The block-based component of the tool is adapted from NeuroBlock, a block-based programming tool for developing neurofeedback applications [19]. The block-based component of NeuroSquare was developed using Blockly. A WebGL stage panel (see Fig. 3 top left) was developed to provide graphical neurofeedback [26]. The tool allows you to create hybrid variables that appear on both the flow and block based environments. This allows users to perform signal processing in the flow-based environments and use that information to drive logic in block-based environments.

D. Hybrid variables

Creating a hybrid variable creates a flow-based box with an input socket and a block-based variable that returns this data. The hybrid variable is updated in the block-based component every time it is evaluated by the flow-based component.

V. Discussion

Our position is that different visual programming paradigms suit different purposes better and by analyzing a specific use-case, we can develop systems that combine the advantages of multiple programming paradigms in a dual-modality system. Studies on NeuroBlock suggest that, while students gained confidence in BCI programming, the small array of affective states (alpha, beta, and engagement) was a drawback [7]. We hypothesize that the flow-based signal processing environment will mitigate this issue by allowing students to calculate their preferred states. The higher flexibility of NeuroSquare may make students more confident with BCI. Furthermore, skills learned with NeuroSquare's flow-based interface may prove useful once users move on to more advanced environments (e.g. OpenVibe, BCILAB, BCI2000).

VI. Limitations

One of the potential limitations to this tool is the transition from the flow-based environment to the block-based environment. While students may find this hard to grasp due to the concealed flow of data, this may also provide insights on how students relate physiological activity and dataflow. Knowledge gained through evaluating these systems could inform ways to design better tools for physiological computing education. Although NeuroSquare only provides acquisition boxes for Muse and OpenBCI Ganglion at this time, the application can be easily extended to any headsets that supports Bluetooth.

VII. Acknowledgements

This work was supported in part by a National Science Foundation (NSF) grant (#1838815).

REFERENCES

[1] M. M. McGill, "Learning to program with personal robots: Influences on student motivation," *Trans. Comput. Educ.*, vol. 12, no. 1, pp. 4:1–4:32, Mar. 2012. [Online]. Available: http://doi.acm.org/10.1145/2133797.2133801

[2] I. Greenberg, D. Kumar, and D. Xu, "Creative coding and visual portfolios for CS1," in *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education (SIGCSE '12)*, pp. 247–252, 2012. [Online]. Available: http://doi.acm.org/10.1145/2157136.2157214

[3] R. H. Sloan, C. Taylor, and R. Warner, "Initial experiences with a CS + Law introduction to Computer Science (CS 1)," in *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '17)*, pp. 40–45. [Online]. Available: http://doi.acm.org/10.1145/3059009.3059029

[4] J. B. Cushing, R. Weiss, and Y. Moritani, "CS0++ broadening computer science at the entry level: Interdisciplinary Science and Computer Science," *J. Comput. Sci. Coll.*, vol. 23, no. 2, pp. 51–57, Dec. 2007.[Online]. Available: http://dl.acm.org/citation.cfm?id=1292428.1292438

[5] Z. Dodds, R. Libeskind-Hadas, and E. Bush, "When CS 1 is Biology 1: Crossdisciplinary collaboration as CS context," in *Proceedings of the Fifteenth Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE '10)*, pp. 219–223. [Online]. Available: http://doi.acm.org/10.1145/1822090.1822152

[6] ——, "Bio1 as CS1: evaluating a crossdisciplinary CS context," in *Proceedings of the 17th ACM annual Conference on Innovation and Technology in Computer Science Education (ITiCSE '12)*, pp. 268–272, 2012.

[7] C. S. Crawford, C. Gardner-McCune, and J. E. Gilbert, "Brain-computer interface for novice programmers," in *Proceedings of the 49th ACM Technical Symposium on Computer Science Education (SIGCSE '18)*, pp. 32–37, 2018.

[8] G. Schalk, D. J. McFarland, T. Hinterberger, N. Birbaumer, and J. R. Wolpaw, "BCI2000: a general-purpose brain-computer interface (BCI) system," *IEEE Transactions on Biomedical Engineering*, vol. 51, no. 6, pp. 1034–1043, 2004.

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

[10] C. A. Kothe and S. Makeig, "BCILAB: A platform for brain–computer interface development," *Journal of Neural Engineering*, vol. 10, no. 5, p. 056014, 2013.

[11] T. R. G. Green, "Conditional program statements and their comprehensibility to professional programmers," *Journal of Occupational Psychology*, vol. 50, no. 2, pp. 93–109. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.2044-8325.1977.tb00363.x

[12] T. Green, R. Bellamy, and J. Parker, "Parsing and gnisrap: a model of device use," in *Human Computer Interaction INTERACT '87*, H.-J. Bullinger and B. Shackel, Eds. Amsterdam: North-Holland, 1987, pp. 65–70. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780444703040500200

[13] I. Vessey and D. Galletta, "Cognitive fit: An empirical study of information acquisition," *Information Systems Research*, vol. 2, no. 1, pp. 63–84, 1991. [Online]. Available: https://doi.org/10.1287/isre.2.1.63

[14] A. Sinha and I. Vessey, "Cognitive fit in recursion and iteration: An empirical study," *IEEE Transactions on Software Engineering*, vol. 18, no. 5, pp. 386–379, 1992.

[15] T. R. G. Green and M. Petre, "Usability analysis of visual programming environments: a cognitive dimensions framework," *Journal of Visual Languages & Computing*, vol. 7, no. 2, pp. 131–174, 1996.

[16] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. S. Silver, B. Silverman, and Y. Kafai, "Scratch: Programming for all." *Commun. ACM*, vol. 52, no. 11, pp. 60–67, 2009.

[17] J. Mönig, Y. Ohshima, and J. Maloney, "Blocks at your fingertips: Blurring the line between blocks and text in GP," in *2015 IEEE Blocks and Beyond Workshop*, pp. 51–53.

[18] D. Weintrop and U. Wilensky, "Comparing block-based and text-based programming in high school computer science classrooms," *ACM Trans. Comput. Educ.*, vol. 18, no. 1, pp. 3:1–3:25, Oct. 2017. [Online]. Available: http://doi.acm.org/10.1145/3089799

[19] C. S. Crawford and J. E. Gilbert, "Neuroblock: A block-based programming approach to neurofeedback application development," in *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC '17)*, pp. 303–307.

[20] D. Weintrop and U. Wilensky, "Between a block and a typeface: Designing and evaluating hybrid programming environments," in *Proceedings of the 2017 Conference on Interaction Design and Children*. ACM, 2017, pp. 183–192.

[21] X. Gao, D. Xu, M. Cheng, and S. Gao, "A BCI-based environmental controller for the motion-disabled," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 11, no. 2, pp. 137–140, 2003.

[22] B. Hernandez-Cuevas, E. Sawyers, L. Bentley, C. Crawford, and M. An-dujar, "Neurophysiological closed-loop control for competitive multi-brain robot interaction," in *International Conference on Applied Human Factors and Ergonomics*, pp. 141–149, Springer, 2019,

[23] J. R. Wolpaw, N. Birbaumer, D. J. McFarland, G. Pfurtscheller, and T. M. Vaughan, "Brain–computer interfaces for communication and control," *Clinical Neurophysiology*, vol. 113, no. 6, pp. 767–791, 2002.

[24] C. Kelleher and R. Pausch, "Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers," *ACM Comput. Surv.*, vol. 37, no. 2, pp. 83–137, Jun. 2005. [Online]. Available: http://doi.acm.org/10.1145/1089733.1089734

[25] P. Stegman, C. Crawford, and J. Gray, "Webbci: An electroencephalography toolkit built on modern web technologies," in *Augmented Cognition: Intelligent Technologies*, D. D. Schmorrow and C. M. Fidopiastis, Eds. Cham: Springer International Publishing, 2018, pp. 212–221.

[26] LLK, "Virtual machine used to represent, run, and maintain the state of programs for scratch 3.0," https://github.com/LLK/scratch-vm.