Mushu, a free- and open source BCI signal acquisition, written in Python

Bastian Venthur[1] and Benjamin Blankertz[2]

***Abstract*— The following paper describes Mushu, a signal acquisition software for retrieval and online streaming of Electroencephalography (EEG) data. It is written, but not limited, to the needs of Brain Computer Interfacing (BCI). It's main goal is to provide a unified interface to EEG data regardless of the amplifiers used. It runs under all major operating systems, like Windows, Mac OS and Linux, is written in Python and is free- and open source software licensed under the terms of the GNU General Public License.**

I. INTRODUCTION

When doing BCI experiments with EEG data one always has to use EEG amplifiers to acquire the brain signals. Those amplifiers usually come with their own software and different vendors have different formats for saving and online streaming of the EEG data. Often the software provided by the amplifier vendors only runs on Microsoft Windows systems, leaving out the Mac OS and Linux users. With Mushu we want to solve those problems altogether: we want a signal acquisition software that runs on all major operating systems, that supports a wide range of EEG hardware, that produces and outputs data in a standardized format independent from the amplifier used and that is free- and open source software.

We decided to use Python as the programming language of choice because we think it is an excellent general purpose programming language with a large and comprehensive standard library. Together with SciPy [8], NumPy [7] and matplotlib [9], Python is a powerful and free alternative to commercial packages like Matlab. Studies [2] have shown that programming in high level languages like Python significantly shortens the amount of time needed to implement a solution and leads to shorter and thus less error-prone code compared to more low level languages like C or C++. This is particularly important for software which is going to be used and modified not only by computer scientists, but students and researchers from different fields.

Figure 1 shows an overview over the general structure of a BCI system. EEG data is measured via EEG caps from the subjects head and the signal is amplified through the EEG amplifiers. A signal acquisition software collects the data from the amplifier and forwards it to the signal processing where usually some machine learning algorithm extracts information from the EEG data and forwards it to the feedback/stimulus presentation. With Pyff [1] we already provided a Pythonic software framework for feedback and stimulus presentation, with Mushu we are targeting now the signal acquisition part. It is the next step in our ongoing effort to provide a completely free- and open source BCI system written in Python.

Similar projects to create complete BCI systems exist, like BCI2000 [5] or OpenVibe [6]. BCI2000 is free only for non-commercial and educational usage and OpenVibe being truly free software is licensed under the terms of the GPL. Both are written in C/C++ which gives better performance, but makes it also much harder to write BCI experiments and applications for non-computer scientists. OpenVibe mitigates this by allowing for an drag- and drop like approach for visual programming of experiments.

Mushu does not try to compete with the commercial software provided by EEG hardware manufacturers which provide much more functionality beyond the scope of Mushu. Brain Products for example sells commercial software like the BrainVision Recorder and BrainVision Analyzer but also provides an open-source software called PyCorder which is a slim signal acquisition software. We believe that hardware manufacturers will benefit if their amplifiers are supported directly by a diversity of software acquisition systems like Mushu.

The rest of the paper is devided into two parts: the first part describes how we reverse-engineered the g.USBamp protocol to write a platform independent driver for it, the second part describes the design of the signal acquisition software.

II. REVERSE-ENGINEERING THE G.USBAMP USB PROTOCOL

The first type of amplifier we implemented a driver for Mushu was the g.USBamp. Although in this paper we elaborate a bit more detailed on how we reverse-engineered the g.USBamp it is important to remember that Mushu is not limited to g.USBamp. One design goal is specifically to

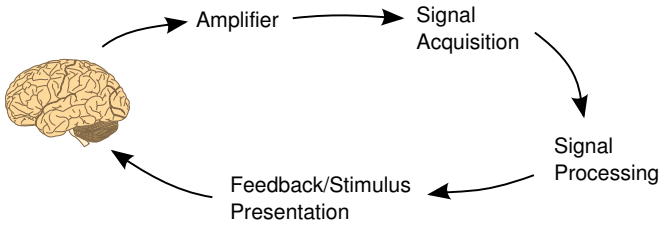

Fig. 1. General structure of a closed loop BCI system. EEG data is acquired from the subject, fed through the amplifier and collected by the signal acquisition. The signal acquisition forwards the EEG data to the signal processing where information is extracted from the data and forwarded to the feedback/stimulus presentation.

[1]B. Venthur, Berlin Institute of Technology, Franklinstr. 28/29, 10587 Berlin, Germany bastian.venthur at tu-berlin.de

[2]B. Blankertz, Berlin Institute of Technology, Franklinstr. 28/29, 10587 Berlin, Germany benjamin.blankertz at tu-berlin.de

support a wide range of EEG amplifiers and provide a unified interface for them.

The g.USBamp is an EEG amplifier produced by g.tec medical engineering GmbH that is widely used in the BCI community. The amplifier is connected to the computer via USB and can record up to 16 EEG channels per amplifier. To record more than 16 channels, several amplifiers can be connected with a synchronization cable. g.tec provides an API and compiled libraries for Windows and Linux allowing to write own signal acquisition software. Unfortunately those drivers have to be purchased by g.tec and cannot be redistributed along with the written software, which make those libraries unsuitable for free software. So we decided to reverse engineer the USB protocol between the amplifier and the computer and implement our own platform independent and free driver.

A. Setup

In order to observe the communication between the amplifier and the computer we connected the amplifier to a Linux computer and ran a Windows XP instance inside a virtualbox environment on that very same computer. Virtualbox is a virtualization software allowing to run an abitrary operating system (OS), like Windows XP, as a guest on a different host OS, like Linux. This happens transparently for the guest OS, which does not notice that it is running in a virtual environment instead of real hardware.

Inside the Windows environment we used the g.USBamp demo tool provided by g.tec which allows for reading the measured data and modifying the various settings on the amplifier. For the g.USBamp tool it looked like is was communicating directly with the amplifier, but since the Windows XP system it was running on, itself ran in an virtualized environment, all communication was proxied through the Linux host system.

On the Linux host system we used a special kernel module, usbmon [11] which allows for monitoring USB traffic between the host system and the USB device. Usbmon works analogous to network monitoring tools like tcpdump. With the help of usbmon we were able to monitor all commands and data sent between the amplifier and the g.USBamp tool and thus had everything we needed to reverse-engineer the protocol.

B. Analysis of the USB Data

Without any prior knowledge about the protocol, we had approach the analysis systematically. We divided the data to analyze into two categories: (a) commands sent to the amplifier and (b) data sent by the amplifier.

To analyze the commands sent to the amplifier, we used the g.USBamp tool to modify one setting at a time and recorded the data sent over USB to the amplifier. It turned out that most of the settings made in the g.USBamp tool translated into one USB request sent to the amplifier. Where settings had different options (e.g. setting the sampling frequency) the parameter is either passed as a value to the USB request or inside the buffer which is sent with every request. Decoding the requests was straight forward after we figured basic things like the endianness and word length of the data transferred. Describing all commands in detail in this paper would be excessive and we kindly refer the reader to the `gtec.py` module we wrote, where every command is documented in detail.

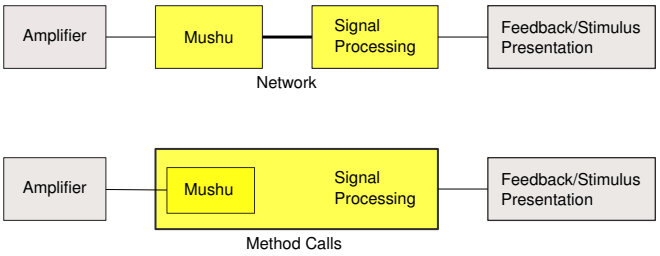

Fig. 2. Two ways of using Mushu: (a) As a stand alone application providing a network interface to the BCI system. (b) directly as a Python library.

Analyzing the data sent by the amplifier was more difficult. We expected to receive packets of fixed length as the number of channels is fixed and the amplifier is supposed to send the measured values for all available channels. Instead we received packets of varying length with random numbers and had no clue how the data was ordered regarding the channels. Fortunately we noticed single zeros within the seemingly random data, appearing at a fixed period. It turned out that those zeros where the values for the trigger line, and from that we could infer the ordering of the rest of the data. It turned out that the amplifier repeatedly sends its data with one measured value per channel: $ch_1, ch_2, \ldots, ch_{17}, ch_1, ch_2 \ldots$. One received packet from the amplifier contains not always exactly 17 (16 channels + trigger line) data points, but sometimes more and sometimes less. One has to buffer incomplete packets and concatenate the next packets, as the amplifier will always send the values in the above order. Moreover, the stream of values has no delimiter whatsoever to mark when all values for a given point in time where transferred, so one has to be careful not do drop any packages as it is not easy to determine which value belongs to which channel just by looking at the raw numbers.

After the analysis of the USB protocol between the g.USBamp and the g.USBamp tool we where able to implement our own driver for the amplifier using PyUSB [10] which provides USB access to the Python language.

III. Design of the Signal Acquisition Software

As we saw in Figure 1, Mushu as a signal acquisition software is placed between the EEG amplifier and the signal processing in the BCI software stack. It reads the raw data from the amplifier, converts it to a standardized format and outputs the data to the signal processing part of the BCI system.

A. Two Use Cases

Mushu's design allows for two different use cases (Figure 2). The first one being the usage of the software directly

as a Python library and the second one being the usage of the software as a stand-alone server providing the EEG data acquired by the amplifier. Using the software as a Python library is convenient when the signal processing part is also written in Python, as one can combine the signal acquisition with the signal processing (and possibly the feedback and stimulus presentation) into a single application.

Using the software as a stand-alone server, decouples the signal acquisition from the rest of the BCI software stack. It is useful if the rest of the BCI system is written in a different programming language than Python, as all partners communicate over network sockets and not via function calls. The server approach makes it also possible to have the different parts of the BCI software stack run on different computers. Functionality-wise are both solutions equivalent and the choice between one of them strongly depends on the connection to the rest of the BCI software stack.

B. Connection to the BCI system

To transfer the acquired EEG data to the later parts of the BCI system, the data is converted into NumPy arrays [7]. NumPy is the de-facto standard for scientific computing in Python as NumPy provides an implementation of fast n-dimensional arrays which is the foundation of other scientific packages as SciPy [8] or matplotlib [9] which provide a free- and open source alternative to commercial packages like Matlab. For each received packet from the amplifier, Mushu translates the data into a matrix where each column is mapped to a channel (i.e. channel 1 $\mapsto$ column 1, etc.) and the rows maintain the ordering of the measured values per channel in time. Having the EEG data stored in NumPy arrays is convenient as NumPy and SciPy use LAPACK [4] and the likes to operate efficiently on those arrays, and efficiency is mandatory for the later signal processing steps.

In the use case where the signal acquisition is used directly as a Python library, the NumPy arrays containing the EEG data are returned directly, in the second use case where the signal acquisition is used as a stand alone application, the arrays are serialized before transmission.

As a second, and more convenient option for a non-Python BCI system, we plan to implement the TOBI interface A [3], an emerging standard for transmission of this kind of bio signals. Implementing the TOBI interface A will allow to connect the Mushu signal acquisition to all BCI systems which comply with the TOBI interface A standard.

IV. CONCLUSION AND OUTLOOK

In this paper we described Mushu, a Pythonic signal acquisition framework for EEG data. Mushu can be used as a Python library or as a stand alone server streaming EEG data in a unified data format from various EEG amplifiers.

We also showed how we reverse-engineered the g.USBamp protocol to write a cross platform driver usable for Mushu. Although Mushu currently only supports the g.tech USBamp amplifier, we are working on supporting other amplifiers by other manufacturers as well, as we want to provide a signal acquisition software which hides the underlying data source through high level commands and thus provides a unified interface for EEG data acquisition for all kinds of EEG hardware.

Mushu is well in the line with our ongoing effort to develop a complete, functional, free and open source BCI system in Python, a powerful and free alternative to commercial software like Matlab, but easier to use for non computer scientists language like C or C++.

Mushu is free- and open source software licensed under the terms of the GNU General Public License (GPL). Mushu's website is: http://bbci.de/mushu.

REFERENCES

[1] Bastian Venthur, Simon Scholler, John Williamson, Sven Dähne, Matthias S Treder, Maria T Kramarek, Klaus-Robert Müller and Benjamin Blankertz. Pyff—A Pythonic Framework for Feedback Applications and Stimulus Presentation in Neuroscience. Frontiers in Neuroscience. 2010. doi: 10.3389/fnins.2010.00179.

[2] L. Prechelt, An empirical comparison of C, C++, Java, Perl, Python, Rexx and Tcl., IEEE Computer, vol. 33, no. 10, pp. 23–29, 2000.

[3] C. Breitwieser, C. Neuper, G.R. Müller-Putz. A Concept to Standardize Raw Biosignal Transmission for Brain-Computer Interfaces. 33rd Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC 11). pp. 6377–6380. Boston, MA. 2011.

[4] Anderson, E. and Bai, Z. and Bischof, C. and Blackford, S. and Demmel, J. and Dongarra, J. and Du Croz, J. and Greenbaum, A. and Hammarling, S. and McKenney, A. and Sorensen, D. LAPACK Users' Guide. Third Edition. Society for Industrial and Applied Mathematics. Philadelphia, PA. 1999. ISBN 0-89871-447-8 (paperback).

[5] 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, 51(6), June 2004.

[6] Y. Renard, F. Lotte, G. Gibert, M. Congedo, E. Maby, V. Delannoy, O. Bertrand, 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, 2010 (in press)

[7] The NumPy project. http://numpy.scipy.org

[8] The SciPy project. http://scipy.org

[9] The matplotlib project. http://matplotlib.sourceforge.net

[10] Tue PyUSB project. http://pyusb.sourceforge.net

[11] Usbmon kernel module documentation. http://www.kernel.org/doc/Documentation/usb/usbmon.txt