IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received September 25, 2020, accepted October 26, 2020, date of publication October 29, 2020,
date of current version November 16, 2020.

*Digital Object Identifier 10.1109/ACCESS.2020.3034770*

# An Easy-to-Use Multi-Source Recording and Synchronization Software for Experimental Trials

**MANUEL MERINO-MONGE, ALBERTO J. MOLINA-CANTERO, JUAN A. CASTRO-GARCÍA, (Graduate Student Member, IEEE), AND ISABEL M. GÓMEZ-GONZÁLEZ, (Senior Member, IEEE)**
Departmento de Tecnología Electrónica, ETSI de Informática, Universidad de Sevilla, 41012 Seville, Spain
Corresponding author: Manuel Merino-Monge (manmermon@dte.us.es)

**ABSTRACT** A typical routine in many scientific studies consists of recording data from devices and identifying which segment of data corresponds with an experimental interval. However, many current applications have been designed to obtain and save data from one single device, and synchronizing data with the markers that delimit the test phases can be difficult. To address this issue, we have developed LSLRec, which is based on Lab-Streaming Layer, a C++ library that allows data synchronization. LSLRec is an easy-to-use, open-source, multi-platform, recording system developed on Java that can save data from several devices at the same time, while maintaining synchronization with the experimental phase markers. It supports three explicit sync methods: the first uses one-integer-channel input Lab-Streaming Layer streams. The others use TCP/UDP socket messages and allow any existing software that generates sync markers through TCP/UDP messages to be used under the same conditions. In LSLRec, the markers are saved together with input data, facilitating the process of linking data with test stages. A prerequisite for recording software is to guarantee suitable timing performance, with no data loss and an easy user interface. These features were assessed for LSLRec, with no data loss detected for 20 hours (25 sessions). We evaluated performance by measuring timestamp deviations for sync marker and input data. The sync methods exhibited an average of deviations of around $-34\mu s$ (which is more than acceptable, e.g., in studies involving living beings), whereas the absolute value of deviation for one-channel data was lower than $40\mu s$. Finally, we assessed the system's usability with the technology acceptance model 3 survey (6 volunteers). Subjects saw the software as useful, and intended to use it in the future. In conclusion, LSLRec is a useful tool for those who need to record data from multiple sources, and maintain synchronization with experimental phases with low delay.

**INDEX TERMS** Recording system, several source devices, lab streaming layer, data synchronization, multi-platform, open source.

## I. INTRODUCTION

Data acquisition and analysis are common tasks in scientific practice, helping us to learn about our environment and improve our interaction with it. A wide range of disciplines study and analyze information from different sources: the estimation of P- and S-wave velocity of seismic waves [10] in geology; the sensing of cell culture [34] in biology; or the analysis of physiological signals for human-computer interaction [13] in engineering. These disciplines all need to use commercial and/or custom-made acquisition devices with application programs to deliver data with different numeric formats, block sizes and sampling rates. It is not always easy for the experimenter to collect data and synchronize them with experimental phases, time intervals or simple events, when several devices and programs are running at the same time, as with multimodal interfaces [20]. To illustrate this, in [26], [27] two different systems were proposed to allow people with disabilities to access a computer. The former modulates the level of attention through an electroencephalography (EEG) sensor, whereas the latter requires residual movements detected by accelerometers or flex sensors. Both systems run with an augmentative-alternative-communication application based on the linear scanning of a panel of ideograms shown on a computer screen. This type of application delivers an event when a new

The associate editor coordinating the review of this manuscript and approving it for publication was Jenny Mahoney.

ideogram is highlighted. Data and events from sensors and the application have to be synchronized to delimit the period of time associated to each highlighted ideogram. A similar situation is described in [37] where authors searched for identified emotional states, through EEG signals, with subjects having to watch affective pictures for several seconds [41]. Every segment of collected data also had to be associated to each projected image. Other examples include handling a wheelchair [2], moving the screen cursor [36] or typewriting [21], [25]. In short, the researcher needs a platform capable of simultaneously recording data coming from multiple sources and accurately aligning the acquired data segments with the experiment phases.

### A. RECORDER SYSTEMS - STATE OF ART

Information sources and their essences are varied. For bioelectrical signals, a reliable recording software is necessary in addition to the appropriate hardware. A wide range of systems exists for harvesting data [5]. Table 1 contains some examples of recording systems.

#### 1) BCI2000

One consolidated system is BCI2000: a general-purpose software designed to record, process, and produce control events from biosignals, focusing particularly on brain-computer interfaces (BCI) [39]. This software has real-time capabilities, guaranteeing no data loss, the use of markers based on User Datagram Protocol (UDP) for data segmentation, and the possibility of adding other tests or forms of computer interaction. However, its usefulness is limited because it does not support multiple acquisition devices during experimental tests, the number of compatible devices is limited and it is not easy to program new applications for BCI2000, and the code reuse is limited.

#### 2) BCI++

BCI++ [33] is a free and open-source framework based on C++, for data recording and the development of BCI applications. It consists of two modules that communicate via Transmission Control Protocol (TCP): a graphic user interface (to create and manage procedures based on 2D/3D graphic engine) and a hardware module for data acquisition, storage, visualization, and real-time processing. The latter allows the acquisition of up to 255 signals from a single device chosen from a compatible hardware list, so, it is not possible to simultaneously record data from two or more devices [23]. Moreover, there is currently no explicit mechanism capable of marking different experimental stages.

#### 3) LAB-STREAMING LAYER AND LAB RECORDER

Lab-Streaming Layer (LSL) takes a different approach to the previous systems. It separates the data acquisition platform from the graphical user interface. According to their authors, LSL is "*a system for the unified collection of measurement time series in research experiments that handles both the networking, time-synchronization, (near-) real-time access as well as optionally the centralized collection, viewing and disk recording of the data*".[1] LSL is a multi-plaftorm library (Windows/Linux/MacOS, and 32/64 bits of computer architecture) that provides a unified interface to record data with several different programming language interfaces (C/C++, C#, Python, Matlab, and Java) with a broad set of supported devices (Emotiv [12], g.USBamp, g.HIamp, Tobii [16], Kinect [45], Serial port inputs, etc.) and allowing the use of multiple acquisition devices at the same time.

LSL stores data in a customized-length network buffer (360 seconds by default or 36-thousand samples for irregular rate). The buffered data are then transferred using local network multicast destination Internet Protocol (IP) and TCP protocols, so the sources and the recording system have to be connected to the same local network, since this IP forces a network without middle routers. The TCP protocol guarantees a reliable, ordered, error-checked delivery and retransmission of lost/failed packets. Hence, data loss is only caused by the LSL's buffer overflow. However, its reliability limits usefulness for real-time applications because of network latency (delay in data reception) and a more complex protocol header which causes longer processing time.

Each device connected to the LSL cloud uses a dedicated stream through which data flow and are temporally buffered along with a timestamp, enabling the synchronization of multiple input streams [17], [38] with a timing error of less than 1 ms. This time synchronization is performed through a local high-resolution computer clock[2] and by measuring the clock offset using the network time protocol (NTP), which estimates the round-trip-time (RTT) between senders and collectors. By adding RTT to every sample timestamp the communication delay is corrected.

LSL designers have developed a C++ software to record data from LSL streams: Lab Recorder (LR).[3] However, the application has limited use, because the number of possible user actions is limited: select/unselect streams, start/stop recording session, and set the output file path. Additionally, the user has to select several live LSL streams, start and stop the recording manually, and all the streams (data and timestamps) are saved in the same file with the extensible data format (XDF) format developed by NASA [31]. Furthermore, the different experimental phases must be demarcated using a specific, separated stream, such that its timestamps delimit the data for a test stage.

#### 4) OPENSIGNALS (r)EVOLUTION

Another system based on the same perspective as described in the section above is OpenSignals (r)evolution (ORE).[4] ORE

[1] Developed by Christian Kothe from the Swartz Center for Computational Neuroscience, Institute for Neural Computation, University of California San Diego, USA. Code available at: `https://github.com/sccn/labstreaminglayer`[Accessed on March 08, 2019]

[2] The developers anticipate that the clock's resolution is better than a millisecond on PC hardware.

[3] Available in May, 2020, version 1.13.0: `ftp://sccn.ucsd.edu/pub/software/LSL/Apps/`

[4] Available in September, 2019: `https://bitalino.com/en/software`

**TABLE 1. Recording Systems. Symbol meanings: OSH - Officially supported hardware; SHC - Supported hardware by the community; NPD - Number of paralleled devices from which data is simultaneously recorded; TS - The total number of signals could be simultaneously recorded from all connected devices; $^u$ unstable devices; ⊗ adding new hardware is not possible; ⊘ unknown; ∞ no limit; * username-password required; § available precompiled executable.**

| System | OSH | SHC | NPD | TS | Marker stage | Libre software | Platforms |
|---|---|---|---|---|---|---|---|
| BCI2000 | 4 | 19 | 1 | ⊘ | local UDP | True* | Windows§, OS X, and Linux |
| BCI++ | 4 | ⊘ | 1 | 255 | No explicit mechanism | True* | Windows§ |
| LabRecorder | >31 | >10 | ∞ | ⊘ | No explicit mechanism | True | Windows§, OS X§, and Linux§ |
| OpenSignals (r)evolution | 4 | ⊗ | 3 | 18 | Add-on | False | Windows§, OS X§ and Linux§ |
| OpenVibe | $19 + 9^u$ | ⊘ | 1 | ⊘ | LSL / From device | True | Windows§ and Linux |

allows real-time data processing, visualization, synchronous recording of up to 3 simultaneous devices, data storing with an output file format based on ASCII text or HDF5 [9], and including LSL as data server. However, the usefulness of this software is limited because the list of compatible hardware only includes 4 devices, the source code is not available and a Bluetooth connection between LSL and the device is required.

#### 5) OpenViBE

OpenViBE is another well-known recording, open-source software that separates the data acquisition server from the client application, allowing sync markers between them [22]. It supports a wide variety of hardware and applications and can include LSL streams since OpenViBE version 2.2.0 [5] as an active device. However, using LSL has several limitations: two streams maximum (data and marker sources) and the numeric format of data and marker are limited to 32-bit float and integer respectively.

There are examples in research where the use of multiple devices for data acquisition is necessary. For example, in the affective computing (AC) field some features must be obtained from several signals [29] like EEG, electrocardiogram, skin temperature, electrodermal activity [8], facial expressions [19], speech audio [35], etc., and using an ad-hoc user interface for eliciting emotions. Only ORE and LR would allow multiple devices operating at the same time but their use for a study like AC, due to their restrictions, would be very difficult to implement.

A key point in any study is deciding how to obtain the data for subsequent analysis. One of the main contributions of this work is a multi-platform, open-source software that provides researchers with a reliable tool for synchronizing and storing data from the equipment used in the lab. This allows the researcher to concentrate on the experimental protocol rather than getting sidetracked by data collection issues. It is also important to highlight that this software combines the marking of data segments with data acquisition of multiple devices in (near-) real time. Other features of this software include: synchronization is compatible with existing software based on signaling through TCP/UCP text messages; data encryption is available; the start/stop recording process can be automated; and the maximum input data rate is also estimated.

[5]Available in May, 2020; `http://openvibe.inria.fr/`

The rest of this article is structured as follows. Design details are given in the Section II, and we then assess its capabilities in Sections III and IV. Its comparison with the above-mentioned systems was ruled out due to their restrictions (synchronization method and/or the limitation in capturing data from multiple sources). Section V contains the discussion of the results and a comparison between LR and our system, and conclusions are presented in Section VI.

## II. LSL RECORDER DESIGN

LSRec[6] is a multi-thread, cross-platform software (Figure 1), based on a previous work,[7] developed using Java technology (version 1.7).[8] Its main features include:

1) *Real-time capacity*: Any data acquisition systems must guarantee an adequate response time to avoid data loss and high latency. LSRec's performance depends on three factors: its design, the capacity of LSL and the operative system. The design of LSRec is based on modules, creating independent non-blocking threads for each one, which minimizes the communication time among them. In addition, a module to estimate the network communication delay and delimit different experimental stages is implemented to minimize the error of marker alignment. Data acquisition depends on LSL's performance, that contains a data collection process that runs in almost real-time. The operative system determines the performance in data's writing actions to hard disk when new data become available.
2) *Scalability*: Number of streams, their channels, sampling rates, or sync markers are not limited by LSRec design, but by the capacities of the Java virtual machine,[9] the operating system and/or the hardware.

LSRec works as follows: Firstly, the experimenter must select the signal streams and the synchronization method that will link data segments to each experimental stage before starting the recording. Then, LSRec creates N+1 threads that store the incoming data from the N selected streams and sync input markers. Any external user application can send

[6]Code available in May, 2019: `https://github.com/manmermon/LSLRecorder`

[7]`https://github.com/manmermon/CLIS`

[8] Dependencies: Java Swing as user interface, JFreeChart (version 1.0.19) and JCommon (version 1.0.23) to plot data, LSL (version 1.13.0), Java Native Access (version 5.1 - LSL's requirement), ICMP4J (version 1020) to estimate the RTT, and Apache Commons Compress (version 1.19) to save data.

[9]Java limitations: slower and more memory-consuming than other programming languages such as C/C++, and garbage collector limits the performance because all threads are stopped to allow this one to be executed.

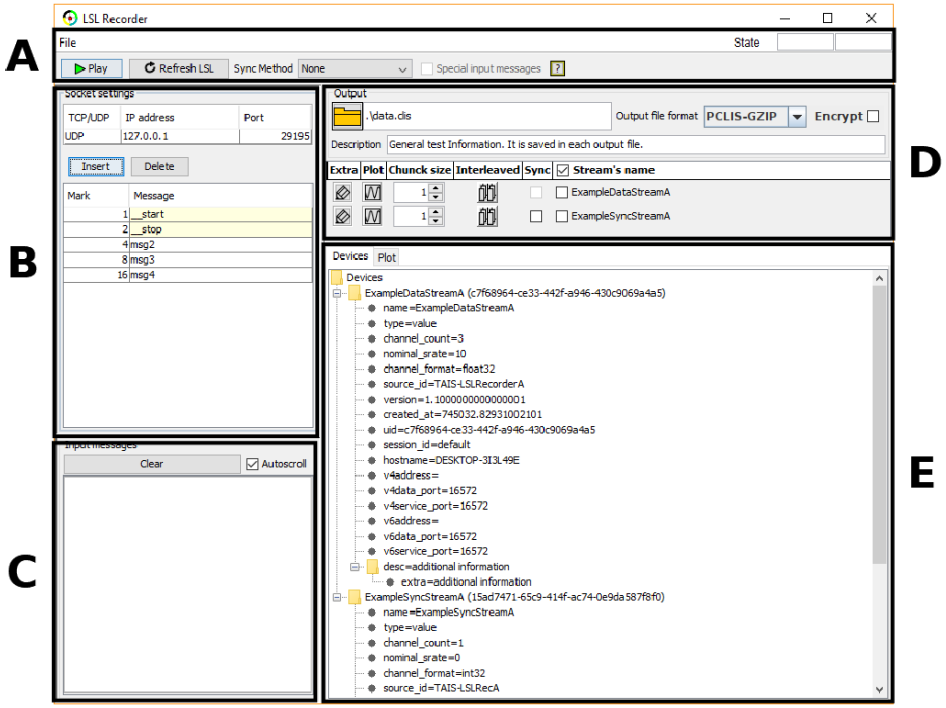


**FIGURE 1.** LSRec 's user interface. A) Control region. Its main elements are: start / stop recording, refresh input LSL streams, selection of sync method to link data and experimental stages, activate special messages to handle the start/stop of recording session by input messages, and recording state. B) Socket settings when sync markers come from TCP/UDP text messages. The user has to register the expected input text messages, and select TCP or UDP, IP address and port. C) Log with received input sync messages. D) LSL-stream settings: output data format, text with general description to save with data, adding extra information in a specific input stream to save jointly with data, encrypting data, activating the plotting of a stream, establishing chunk size and interleaving process, and selection of data or sync-marker streams. E) LSL-stream details (to save jointly with input data) and input data plot to check they are correct.

sync markers either through a specific LSL stream or via text messages over a socket connection. When the trial finishes and data are aligned to sync marker, data files are finally compressed.

LSRec has five different components based on modular architecture and component-based design methodology (Figure 2): main coordinator, streaming data supervisor, synchronization thread, data receivers, and encoders. These are based on the SOLID principle [32]: single responsibility, open/closed, Liscov substitution, interface segregation, and dependency inversion.

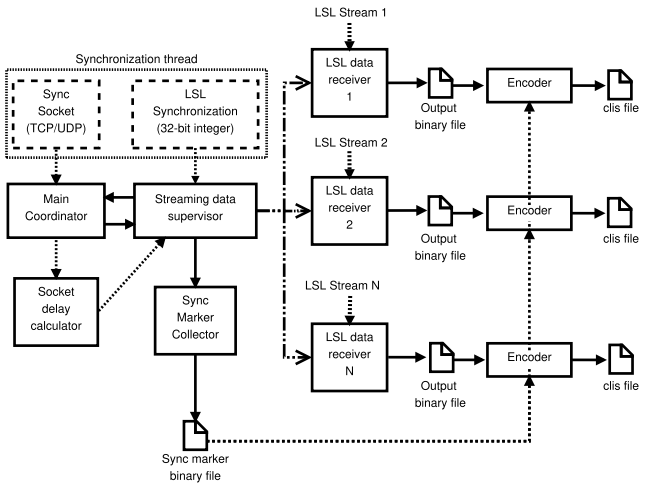


**FIGURE 2.** LSRec design. Five main components: main coordinator to start/stop recording, data-stream supervisor to save input data through receiver threads, synchronization thread to register the experimental-stage markers, data receivers, and output format encoders.

### A. MAIN COORDINATOR

This component is the general system manager, and its main tasks are to check the settings, launch/stop the recording process, and report an estimation of the communication network delay to the streaming data supervisor (SDS) by redirecting the sync markers received from registered input socket messages.

### B. STREAMING DATA SUPERVISOR

This element creates all data receiver threads from the user's selected LSL streams (one for each selected stream), and another single thread to collect the received sync markers from *synchronization threads* (Section II-C) and reports the state of data receiver thread to the *main coordinator*. In addition, when the data recording is stopped, this supervisor initiates the encoder threads to transform the temporary binary data files to final compressed output files.

### C. SYNCHRONIZATION THREAD

Every single LSL stream is formed by samples and timestamps. The timestamps are obtained from the clock of the system where LSL is running. In the same way, sync markers also have timestamps. Data segmentation is performed by comparing temporal points in sync markers to data stream timestamps. LSRec does this operation for the experimenter by automatically interleaving data and sync markers, generating a unique output data set with both of them.

In LSRec, the sync markers and their timestamps are stored in a single independent binary file that is employed in the encoders to delimit the different test stages (Section II-E). The markers can be either TCP/UDP socket text messages (*sync socket*) or one or more 32-bit-integer LSL streams from a single channel (*LSL synchronization*) (Figure 2). In the first option, LSRec acts as a server without any client limitation with the experimenter having to register all expected messages. In this process, LSRec assigns a sync marker, a 32-bit integer with a value that is a power of two, to each message. During the operation, when the *sync socket* receives a message, this is firstly redirected to the *main coordinator* that checks if it is registered. If so, its associated sync marker is then sent to SDS jointly with its timestamp. Note that the input socket message must end with the new-line character (\n). In the second option, *LSL synchronization*, the user must select one or more LSL input sync streams. Each received integer value is considered as a sync marker, and sent straight to SDS. In this strategy, unlike *sync socket*, markers are not limited to powers of 2, since these values are not set by LSRec but by the user, who must establish the appropriate values. Note that marker sources are unlimited in both synchronization procedures, making it possible to coordinate two or more independent trial systems.

Each received sync marker from a LSL stream has a timestamp, meaning an additional process to estimate the time the marker was sent is not needed. Instead, the socket message has to calculate its timestamp. This is obtained from *socket delay calculator*. This subtracts the half of RTT from the time the message is received. RTT is the shortest time from 4 type 8 (echo) ICMP messages. The aim of this is a better estimation of the moment the marker was sent.

Unlike other recording software, such as LR, the start and stop actions can be automated using two special sync markers: values 1 and 2. Thus, when the user activates this mode, LSRec starts in standby mode (without recording data). The recording session starts to retain data when sync marker 1 is received (or the message "*__start\ n*" using sync socket), and the recording process ends with sync marker 2 (or *"__stop\n"* message). This leads to better control of the trials.

### D. DATA RECEIVER

Users can select several LSL input data streams, with LSRec creating an independent data receiver thread (DRT) for each one. The input arrays data and timestamps are stored in a temporal binary file as soon as possible. Each LSL stream may be formed by one or more channels where each one, in turn, can contain chunks of several samples, so that a sorting process may be necessary if interleaved data is required. This operation is done in the encoder module to prevent it from affecting the data recording.

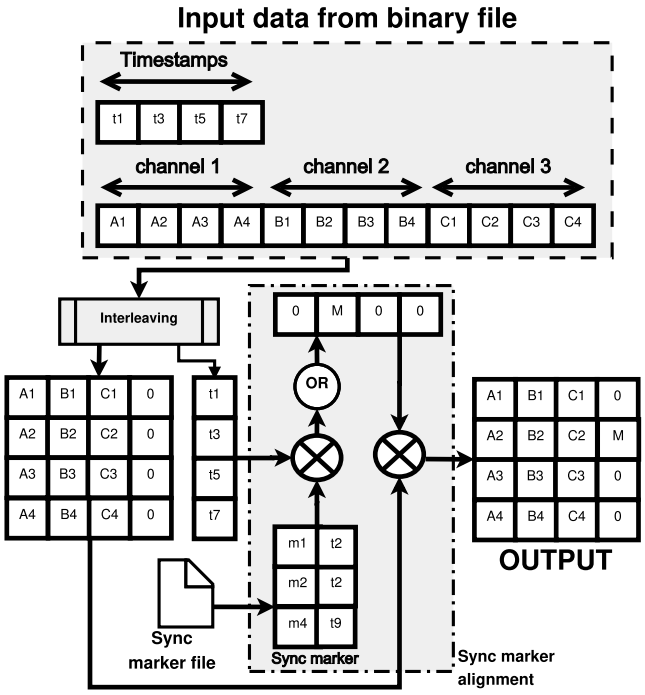


**FIGURE 3.** Interleaving process and sync marker alignment. An example for the input array of 3 channels and chunk size of 4 elements. The marker *M* is the result of a bitwise OR operation between the markers *m1* and *m2*. The input data is firstly shaped as a 2D matrix with interleaved values by rows. The next step is the alignment of sync markers. The marker's timestamps are compared with data timestamps to determine their rows in the data matrix.

Keeping the data in a volatile memory may lead to their loss due to an accidental shutdown of the computer/software or an out of memory error. The temporary output binary files have been designed to overcome this problem by directly storing the data on the hard disk. Note that the real-time capacity of LSRec is limited by the operating system, which determines the performance in data's writing actions into hard disk as new data become available. LSRec's *File* menu includes an option to assess these writing actions for each selected input stream, allowing the estimation of the maximum data rate. If any of these estimated rates are smaller than the input sampling rate, there could be data loss due to an overflow in the LSL's buffers.

LSRec only collects data from different streams, while the other software take responsibility for setting and running the LSL streams to send data. The Appendix A shows an example code of the process to create an output data stream.

### E. ENCODERS

As the recording process finishes, an encoder thread (one for each stream) generates an output file. The encoder performs three processes: data interleaving, aligning of data with sync

markers, and storing data in the output file. The first two tasks are sketched in the Figure 3.

The interleaving process is performed if the user selects this option before starting the recording. LSRec handles data as a matrix, where the columns are the channels and the rows contain the received data, and this presupposes that they are interleaved in the input array with respect to channels, if not, they need sorting. For example, an LSL stream of 3 channels and chunk size of 4 produces an input array that has 12 elements (Figure 3). The first, fourth, seventh, and tenth positions belong to channel 1; the second, fifth, eighth, and eleventh locations corresponds to channel 2 data; and the rest belong to channel 3 data. These are sorted as a $4 \times 3$ matrix, in which a column for the sync markers (CSM) is added. The default value for each cell of this new column is zero, which means that there has been no change in the trial since the previous sync marker was received.

The next step is to align data with the sync markers, which have been placed in chronological order beforehand. They are sorted in 5-mebibyte blocks. Each sync marker is inserted in the CSM, in the first row that has a greater or equal timestamp. If several sync markers are assigned to the same row, a bitwise OR operation takes place. For this reason, each bit in a sync marker is considered as a flag. It should be noted that the input integer markers with *LSL synchronization* are not restricted to the power of 2, so the user has to set the correct values bearing in mind the bitwise OR operation.

A final aspect is to reduce the space transferring the data to disk for which there are many different compression techniques. One with a good compression ratio is the GZIP algorithm [18], which receives a byte array and returns another much shorter one. However, memory problems may arise if the input binary array is too long. To address this issue, the compression technique can be applied to segments of shorter lengths, with data being fragmented into smaller pieces. In LSRec, the input binary data file is read byte by byte and moved into a 10 mebibytes (10MiB) queue (this size is based on Winrar software[10]). The data matrix is queued item by item according to row order. When the queue is full, data is compressed, stored in the output file, and the queue emptied. This process is repeated until all bytes are read. The Appendix B describes a template of the encoder process.

### F. SIGNALS

LSL was designed to record signals at uniform or nonuniform sampling rates. Therefore, LSRec, which is based on LSL can record signals attained at regular rate (electroencephalogram, electromyogram, skin temperature, audio, video, etc.) or asynchronously (computer-mouse-pointer movements, keystrokes, events like turning on/off a light switch, etc). An example of this is shown in [7] (chapter 7), where participants in the experiment took part in several game sessions using Kinect and the application registered 4 signals at a regular sampling rate (electrocardiogram, two channels of electrodermal activity, and the skin temperature) and another coming from Kinect (skeleton points) and the game events. All of them were at an irregular rate.

To decrease the possibility of data loss, the LSL's network buffer length is set to 100 million samples when a sampling rate is undefined; otherwise, such a length is limited up to 360 seconds for each channel in the input stream. For example, the buffer size of a 10-Hz-sampling-rate input stream with 3 channels will be of 10800 samples ($360 \times 10 \times 3$).

### G. SYSTEM REQUIREMENTS

LSRec has been designed using Java 7 and testing on Windows 10. Java Development Kit (JDK) 8 or greater must be installed for LSRec to work correctly, and 20-MB free disk space is required along with LSL's dll file. Furthermore, LSRec requires a minimum of 80 MB RAM memory space and fulfills the requirements of the operative system itself, these include a Pentium 2 266 MHz processor, with at least 450 MB of free hard disk and 128-MB RAM memory.

### H. USE CASE EXAMPLES

LSRec is a general-purpose recorder system with different options to make it easier to capture and analyze experimental data. These options are described below.

#### 1) ADDING EXPERIMENTAL DETAILS

Different tests can be performed in the same study and LSRec allows details to be added like, for example, difficulty level during the analysis of the cognitive load from arithmetic activities [24], [40], or certain device settings such as the value of the reference resistance [6], which may change among subjects, when transforming voltage to conductance in electrodermal activity [3]. Thus, extra information about experimental settings is sometimes necessary for further data analysis. General test details and specific device information are written down in the *Description* field and stream's *extra* button in area *D* in Figure 1. The experimental *description* is inserted in all input LSL streams, whereas the *extra* details of an input data source are only added in that specific stream. Thus, each output file contains all the information required to conduct the analysis successfully.

#### 2) DATA ENCRYPTION

One important aspect to be considered when experimenting with people is data privacy. The collected information must be protected against external agents and, for this reason, LSRec supports data encryption (section C). The experimenter can select the encrypt box and start data recording (*Encrypt* checkbox in the area *D* in the Figure 1). A pop-up window will appear to apply for the key that will be used to encrypt the data in the encoders (section II-E).

#### 3) SETTING SYNC METHODS

LSRec implements different *sync mechanics* for signaling experimental stages and input data. The user has to select the

[10] Available in March, 2019, version 5.50: `https://www.winrar.es/`

method to be used in the trial (Figure 1, area *A*): socket and LSL streams. The former setting consists of: transport layer protocol (TCP/UDP), IP address (not restricted to localhost address), port number (from 1025 up to 65535), and expected input messages. When LSL streams are selected, the socket setting is ignored and the user has to choose which input streams emit the sync markers (*Sync* checkboxes in the area *D* in the Figure 1). Note the sync streams cannot be selected as data source (the last column in the area *D* of the Figure 1), and these are only available in 32-bit-integer streams of one single channel.

#### 4) STREAMS' SETTINGS

The stream setting consists of: the data chunk size, activating the interleaved data during the encoding process, and selecting the input data stream (or sync). LSRec has not implemented any automatic process to find new devices. The user must execute this action by clicking the button *Refresh LSL*.

#### 5) CHECKING INPUT STREAMS

Some devices require the input data to be checked beforehand to prevent a test from failing. For example, when using wearable systems, a noisy signal can be recorded due to low battery. Each input stream has a *plot* button, such that each channel is drawn in an independent figure. This way, clicking on this button allows stream settings and data quality to be checked.

#### 6) START/STOP RECORD

The recording process starts/stops normally by clicking the *Play/Stop* button (Figure 1, the area *A*). Another alternative is utilized by the *Special input messages*. When this function is activated, data are recorded after a *start* message is received. In the same way, the reception of a *stop* message terminates the recording process. These messages have been indicated in the Section II-C.

#### 7) ESTIMATION OF MAXIMUM INPUT DATA RATE

The operating system determines the performance in data's writing actions into hard disk as new data become available. If the time spent performing these actions is less than the input sampling period, there will be no missing data. LSRec allows users to assess these times to avoid data loss.

The researcher must select the input streams that will be recorded and then, the option *Writing test*, in the *File* menu (Figure 1, area *A*). Data are recorded for one minute after which the average time values for writing are shown.

## III. SYSTEM TESTING

Three main aspects were explored in the assessment of LSRec: timing performance, percentage of missing samples and software usability. As explained above, each experimental phase must be correctly identified by the sync markers. Minimizing the time deviation between the data and the generation of sync markers is necessary in many studies, otherwise the data analysis could be invalidated. This is the case for experiments involving evoked potentials, such as P300 or N100 [30], in which incorrect signaling influences the averaging of epochs and, consequently, the estimation of the evoked potential itself. We designed the first test in this study to measure these deviations. The software also had to ensure that all data generated and received during the experiment were correctly captured and stored: in other words, that there were no missing data. This was the aim of the second test. A final, but no less important aspect, concerned the usability of the application in terms of how difficult or easy it was for an experimenter to use LSRec. It is important to gauge the degree of acceptance of a specific technology as this has a direct bearing on the success of future implantation.

In the following sections, we give details of the tests conducted to answer these questions. To execute the tests, we used a single PC with 3.00-GHz Intel Core i5-2320 (4 cores, 4 threads, and 6MB of CPU cache), 8 GB RAM, a serial ATA hard disk of up to 156 MB/s of write speed, and 64-bit Windows 10 (Education edition - version 17134.825), and results were analyzed with Matlab 9.0.0.341360 (R2016a).

### A. MEASURING SYNCHRONIZATION DEVIATIONS

LSRec has to ensure that calculated timestamps for sync markers are as close as possible to the time point in which they were generated. The delay was calculated using all sync methods, so that only one of them was activated in each session. The 60-minute recording sessions were repeated 15 times (5 with UDP socket and 5 with TCP socket for a localhost IP address, and 5 using a sync LSL streaming). The rate of sync markers was set to 10 messages per second and the clock's time was sent in a double, one-channel LSL stream to evaluate the marker's delay. At the same time, two independent 30-channel LSL streams of data were recorded. Each one sent a uniform-distribution random signal of float and double data type, with sampling rates set to 256 Hz. We established these parameters to check that the system worked correctly with different kinds of streams.

### B. PERFORMANCE TEST

To measure the performance of LSRec we designed an experiment with five runs (Table 3). The first three runs used three different sampling rates (512, 4k and 16kHz ) and consisted of five 60-min recording sessions in which the system clock was measured. The last two runs consisted of recording a total of ten 30-minute videos with different frame resolutions ($176 \times 144$ and $320 \times 240$ pixels) at 30 frame per second (FPS) where each frame was sent row by row. Additionally, two other data streams were recorded for the five runs: the

**TABLE 2.** Items of technology acceptance model 3 (TAM3) survey.

| | Category | **Definition** from [43] |
|---|---|---|
| | Behavioral intention (BI) | The willingness to adopt the new software. |
| Perceived usefulness | Perceived ease of use (PEOU) | The degree to which a person believes that using an information technologies will be free of effort. |
| | Perceived usefulness (PU) | The extent to which a person believes that using an information technologies will enhance his or her job performance. |
| | Voluntariness (VOL) | "The degree to which use of the innovation is perceived as being voluntary, or of free will" [28]. |
| | Subjective norm (SN) | The degree to which an individual perceives that most people who are important to him think he should or should not use the system. |
| | Job relevance (REL) | The degree to which an individual believes that the target system is applicable to his or her job. |
| | Output quality (OUT) | The degree to which an individual believes that the system performs his or her job tasks well. |
| | Result demonstrability (RES) | The degree to which an individual believes that the results of using a system are tangible, observable, and communicable. |
| Perceived ease of use | Computer self-efficacy (CSE) | The degree to which an individual believes that he or she has the ability to perform a specific task/job using the computer. |
| | Perceptions of external control (PEC) | The degree to which an individual believes that organizational and technical resources exist to support the use of the system. |
| | Computer playfulness (CPLAY) | "[...]the degree of cognitive spontaneity in microcomputer interactions" [44]. |
| | Perceived enjoyment (ENJ) | The extent to which "the activity of using a specific system is perceived to be enjoyable in its own right, aside from any performance consequences resulting from system use". |
| | Objective usability (OU) | A "comparison of systems based on the actual level (rather than perceptions) of effort required to completing specific tasks" [42]. |

**TABLE 3.** Performance measurements.

| Signal | Time (s) | Data | Channels | Chunk size | Sampling rate (Hz) | Timestamp deviation | Data loss (%) | CPU Load (%) |
|---|---|---|---|---|---|---|---|---|
| System clock | 3600 | double | 1 | 1 | 512 | $-7.324$ ($\pm 33.625$) $\mu s$ | 0 | 0.401 ($\pm 0.581$) |
| System clock | 3600 | double | 1 | 1 | 4k | $-10.659$ ($\pm 22.348$) $\mu s$ | 0 | 0.442 ($\pm 0.621$) |
| System clock | 3600 | double | 1 | 1 | 16k | $-13.546$ ($\pm 21.072$) $\mu s$ | 0 | 11.692 ($\pm 9.354$) |
| Video ($176 \times 144$ ) | 1800 | int32 | 176 | 144 | 30 | $-6.143$ ($\pm 13.888$) $ms$ | 0 | 5.194 ($\pm 2.576$) |
| Video ($320 \times 240$) | 1800 | int32 | 320 | 240 | 30 | 21.364 ($\pm 48.179$) $ms$ | 0 | 8.776 ($\pm 2.678$) |

CPU load[11] and the point in time in which a new data was generated (system clock or frame).

### C. COMPRESSION

At the same time as the previous tests, we calculated the compression rate for the Captured Long Input Streams (CLIS) output file (Appendix C) to determine the saved memory space. The worst case possible was evaluated from uniform-distribution random streams, whereas the better cases possible were assessed with signals that show regularity. This is because CLIS' compression is based on the GZip algorithm, which uses Huffman coding. It needs a smaller number of bits to code deterministic signals than random ones. This way, we obtained the general interval of compression rate.

### D. USABILITY SURVEY

Usability surveys assess a system's ability to reach its designed goal [11], measuring ease of use, perceived usefulness, and intention to incorporate it into the production system, etc. For this reason, we evaluated the usefulness of LSRec using TAM3 [43], which provides a clear picture of users' behaviour and their reasons for evaluating a software in terms of usefulness and ease of use. Both features were assessed by scoring (from 1 to 7) system features, social influences, individual differences, subjective norm, image, job importance, etc. The TAM3 test consists of a set of 16 questions, but in this study we used 13 (Table 2). Each subject filled in the questionnaire after completing a task guide with different activities (Appendix E), with a secondary aim: to obtain feedback about the main aspects that need to be included in the future user guide. With this aim in mind, the task guide did not contain any information about how LSRec could be set, and consequently, we removed the questions from the TAM3 survey about help documentation (CSE2 and CSE4). The usability test was filled in voluntarily by 6 researchers in the signal processing field from the Departamento de Tecnología Electrónica of Universidad de Sevilla (Spain). None of them had used LSRec before. Subjects were only given information when they were stuck and did not know how to continue.

### E. UNIT AND INTEGRATION TESTING

LSRec has been developed following the White-box Testing method [1], therefore the software has been assessed to check whether each element (class, method, function) is

[11]The library Sigar was utilized to register the CPU load. Code available at: `https://github.com/hyperic/sigar`[Accessed on January 07, 2020].

**TABLE 4.** Group correlation from TAM3 survey. The symbols mean: *M* is average, and *SD* is standard deviation, * is p-value< 0.05 in the correlation test.

| | M | SD | PU | PEOU | CSE | PEC | CPLAY | ENJ | OU | SN | VOL | REL | OUT | RES | BI |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PU | 6.46 | 0.64 | 1.00 | | | | | | | | | | | | |
| PEOU | 6.33 | 1.11 | 0.17 | 1.00 | | | | | | | | | | | |
| CSE | 6.17 | 1.14 | 0.45 | 0.93* | 1.00 | | | | | | | | | | |
| PEC | 6.33 | 1.37 | 0.21 | 0.95* | 0.91* | 1.00 | | | | | | | | | |
| CPLAY | 5.13 | 1.81 | 0.21 | 0.33 | 0.27 | 0.10 | 1.00 | | | | | | | | |
| ENJ | 5.50 | 1.21 | 0.84* | 0.15 | 0.31 | 0.06 | 0.48 | 1.00 | | | | | | | |
| OU | 4.83 | 2.11 | -0.16 | 0.30 | 0.10 | 0.05 | 0.72 | 0.37 | 1.00 | | | | | | |
| SN | 6.38 | 0.81 | 0.70 | 0.11 | 0.38 | 0.02 | 0.15 | 0.73 | 0.07 | 1.00 | | | | | |
| VOL | 6.78 | 0.53 | 0.28 | 0.62 | 0.53 | 0.61 | 0.63 | 0.25 | 0.28 | -0.26 | 1.00 | | | | |
| REL | 6.50 | 0.83 | 0.33 | -0.02 | 0.08 | -0.22 | 0.25 | 0.66 | 0.50 | 0.82* | -0.39 | 1.00 | | | |
| OUT | 6.50 | 0.96 | 0.07 | 0.52 | 0.47 | 0.30 | 0.44 | 0.45 | 0.76 | 0.55 | -0.00 | 0.78 | 1.00 | | |
| RES | 6.04 | 1.62 | 0.54 | -0.29 | -0.21 | -0.17 | -0.16 | 0.58 | -0.06 | 0.14 | 0.05 | 0.14 | -0.20 | 1.00 | |
| BI | 6.28 | 1.04 | 0.93* | 0.27 | 0.58 | 0.27 | 0.15 | 0.77 | -0.14 | 0.89* | 0.08 | 0.51 | 0.30 | 0.29 | 1.00 |

fully functional: the input parameters and the returned data type and their values have been verified through test cases for each function, so that each one of them is independent of the rest. For example, interleaving function needs three input parameters (Figure 3): data array, number of channels and chuck size. This was tested using different settings: 1) null data array; 2) no-null empty data array; 3) no-null, no-empty data array with number of channels and chunk size equal to 0 and they both equaled -1; 4) no-null, no-empty data array whose length was less than the number of channels; 5) no-null, no-empty data array whose length was greater than number of channels, but less than chunk size; and 6) no-null, no-empty data array whose length was divisible by the number of channels and the chunk size. In integration testing [4], all units were integrated within a program and tested as a group, so that subjects were able to supply annotations to reveal interface failures between the modules/functions. For example, data interleaving process and sync marker alignment were used by encoders to save them in the selected output format file.

## IV. RESULTS

The results of the system testing are described in the two following sections. Previously, we applied the interquartile-range method to remove outlier values $[Q1 - 1.5(Q3 - Q1), Q3 + 1.5(Q3 - Q1)]$, where $Q1$ and $Q3$ represent the lowest and uppermost quartiles respectively.

### A. DATA ACQUISITION

The results of the deviation of sync markers are shown in Figure 4. The central 90% of deviations were congregated in the range of (−0.068, 0.012) ms for UDP, (−0.090, 0.018) ms for TCP, and (−0.035, −0.008) ms for LSL, and the medians were −0.026 ms in UDP, −0.034 ms with TCP, and -0.019 ms for LSL. In general, sync markers timestamps were estimated before the point of time that they were generated. These deviations exhibit a greater spread in TCP, followed by UDP, and finally LSL. A Kruskal-Wallis analysis indicates a statistically significant difference between the three sync methods (p-value< 0.01).

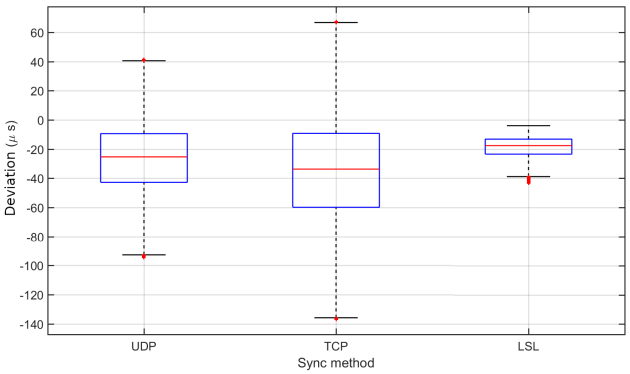


**FIGURE 4.** Estimation of time deviation of sync markers without outliers. The quartiles Q1 and Q3 are both negative values, such that, central 50% of deviations are between -60 and 10 $\mu$s. The interquartiles range are smaller than 35, 60, and 15 $\mu$s for UDP, TCP, and LSL respectively. Tails are not exhibited in data distribution.

The performance tests show no missing data from any of the 25 sessions was detected (Table 3). The data timestamps exhibit an incremental deviation with respect to the amount of data received per second. The lowest deviations were obtained using one-channel streams with chunk size of 1 sample *one-channel stream with chunk size of 1 sample* (1CS), in the $\pm 40\mu$s interval. The deviations for video were calculated subtracting the moment the frame was captured in source software to the LSL timestamp of the last frame's row. For video resolution of 176 × 144 pixels, the time range was ±20 ms, whereas for resolutions of 320×240 pixels, the deviations were from −27 ms up to 70 ms. Moreover, the CPU load was always lower than 20%, so that the approximate average for all tests hovered around 5%.

With respect to the compression rate, the averaged value obtained from random data was 7.26 (±1.97)%, whereas system-clock and video signals were 26.81 (±5.12)% and 52.52 (±7.02)% respectively.

### B. USER USABILITY SURVEY

The analysis of the TAM3 test gauged subjects' intention to use LSRec. The relationships between the survey's questions and categories were analyzed by using the Pearson

correlation coefficient ($\rho$), and we applied the no-correlation hypothesis test. Table 4 shows the survey results by category (the correlation among questions is included in the supplementary materials). The BI category obtained an average of 6.28($\pm$1.04), showing a statistical relationship with PU ($\rho$= 0.93, $p < 0.05$), meaning that subjects intended to use this software because it was perceived as useful (PU's average = 6.46($\pm$0.64)). However, LSRec was also perceived as easy to use (PEOU= 6.33 $\pm$ 1.11), in spite of dependencies with BI or PU were not found. Therefore, the intention to adopt this software can be put down solely to its usefulness.

On the other hand, the perceived usefulness linked indirectly to the features SN and REL ($\rho$= 0.82, $p < 0.05$). The working environment favorable to using the system made subjects assume that it was suitable. Furthermore, PEOU showed significant dependency with CSE ($\rho$= 0.93, $p < 0.05$) and PEC ($\rho$= 0.95, $p < 0.05$) which, in turn, were indirectly related to PEOU ($\rho$= 0.91, $p < 0.05$). The system was rated as easy to use because users felt that they had sufficient knowledge to use it correctly and the system was backed up by their organization.

## V. DISCUSSION

Many scientific studies need to coordinate multiple data sources simultaneously to determine what is happening in an experiment. Thus, a reliable application software is a key factor for collecting data from different devices. LSRec and LR allow good data synchronization from multiple sources because they are both based on LSL. This library puts a timestamp on each sample inserted in an output LSL stream. Our results have shown that the deviation time between timestamps is of $\pm 40\mu$s for a one-channel stream (Table 3), which means that a misalignment of 1 or more samples among different data sources may occur when the sampling rate exceeds 12.5 kHz. The sampling rate is often much lower than 12.5 kHz so that only a few misaligned samples can be found, which is acceptable. For example, in studies concerning evoked potentials, such as P300 or N100 [30], a deviation of 1 ms is not significant. We find another example when comparing heartbeat rhythm from an ECG and the recorded sound from a stethoscope with a sampling rate of 44 kHz for both signals. In this situation, the desynchronization could be around 4 samples, which is not very important. The desynchronization is aggravated in multi-channel streams, where data are captured by chunk lengths greater than 1 sample. For a video with a rate of 30 FPS and resolution of 176 $\times$ 144 pixels, the deviation is between $\pm$20 ms, which means that the desynchronization is higher than 1 frame from 25 FPS. This desynchronization worsens for higher resolution video, to the extent that desynchronization can be detected from 10 FPS with video of 320 $\times$ 240 pixels. Nevertheless, no data is lost, even with a 320-per-240-pixel video with 30 FPS, where it is equivalent to a sampling rate of 2.3 MHz for 1CS. These results allow us to assert that LSL is an efficient tool for recording data from different sources with good synchronization.

LR is a recording software developed by the designers of LSL which is similar to LSRec: so why should experimenters use LSRec instead? LR has several pros but also some cons when compared to LSRec. LR is written in C++, so one would expect a better use of system resources. However, this programming language depends on the hardware architecture and operating system, so the code has to be compiled again whenever the system changes. LSRec is based on Java, which uses a virtual machine that makes the software independent of the system and allows LSRec to be executed on any platform.

The compatibility of LR with existing software is limited because only one synchronization strategy has been implemented and in which the experimental-phase markers must be sent as a separate LSL stream. LSRec supports and extends this synchronization strategy adding TCP/UDP messages, which allows the coordination of the input data with test events from existing experimental applications based on TCP/UDP transmissions. Hence, LSRec facilitates the combination with research software. Furthermore, LSRec allows sync markers from more than one source, so one could use two independent trial systems at the same time. To illustrate this, let us assume we want to assess the effect of light brightness when performing basic math operations. One system would generate a sync marker when a new arithmetic activity was shown, while another would change light brightness and send another different sync marker.

One exclusive option of LSRec, which is not supported by LR, is the ability to automatically start/stop data capture via special input messages, which improves control in experimental trials.

Another drawback with LR is that the data segmentation process needs the timestamps of data and sync markers to be aligned to link them with an experimental phase, and this can be tedious (Figure 5, on the left). LSRec supports this scheme by saving markers as an input data stream, and it also allows the joint recording of each sync marker with the received data array, facilitating the data segmentation process (Section II-E; Figure 5, on the right).

One of the main drawbacks with LR is that it ignores the states of the input streams, so that the researcher may not realize when some of them are inoperative. This feedback is an important point because there has to be a guarantee that data are correctly collected during the experimental session. LSRec stops recording a session when some input streams are closed or when, for streams with regular sampling, no data are received for 3 times the sampling period with an upper limit of 3 seconds, if the sampling rate is higher than 1Hz. This allows the detection of device problems caused by low battery, communication interference, breakdown, etc. Likewise, this timer is disabled for irregular-sampling-rate streams.

Another topic concerns the output files and the use of memory resources. LR directly saves the selected input streams in the same output file with a format based on XDF. Even if we are only interested in analyzing one stream, LR obliges us to load the whole file which may cause memory problems if it is heavy. In contrast, LSRec generates individual files for

**TABLE 5. SWOT analysis applied to the software architecture.**

| | | |
|---|---|---|
| Internal | Strengths | Weakness |
| | • Open source.<br>• Multi-platform software.<br>• Unlimited sampling frequency.<br>• Multiple devices can be connected.<br>• Cooperative multitasking.<br>• No data loss has been observed.<br>• Modular design.<br>• Easy synchronization with research tests.<br>• Multiple sync methods.<br>• Saving markers jointly with data in separate output files ensures no marker loss due to unintentionally removing marker file.<br>• Start/stop recording process can be automated. | • Based uniquely on LSL. No other device libraries are supported.<br>• Data transmission as text strings in socket messages.<br>• Network latency limits its usefulness.<br>• Delay of sync markers.<br>• Own format output file.<br>• Redundant markers when they are jointly saved with data in separate output files. |
| External | Opportunities | Threats |
| | • Open source software which might extend its capabilities.<br>• Multi-platform system produces independence from hardware architecture (x86, x64) and operating system (Windows/Mac/Linux). | • Changes in the LSL library may stop the software from working. |

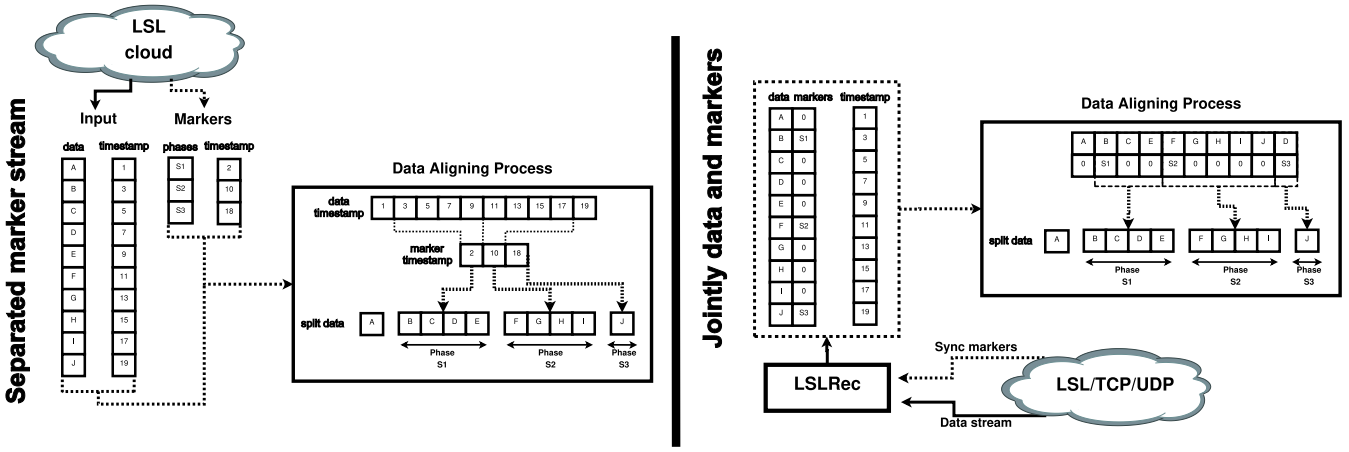


**FIGURE 5. Data segmentation process. On the left, aligning data and markers using timestamps. On the right, segmentation when markers are saved together with input data.**

each stream, thereby reducing memory problems. The LSRec format (CLIS) uses the GZIP algorithm as a compression technique (Appendix C) exhibiting a good compression ratio, reducing memory space by around 7% for random input data (worst case) and 52% for the best case.

Other exclusive options in LSRec include: the possibility of adding extra information to the stream descriptors and a general description about the recording session can be saved in output files; plotting a data stream before the experiment to verify that the devices work correctly; a hard disk write test by selected stream to determine the maximum sampling rate from which data can be lost due to the LSL buffer being filled; and an encoder from the output, temporal binary file to prevent an experimental session being lost due to a problem with the system (shutdown, blocking, etc.).

In addition, the CPU load is generally low when using LSRec (below 12%), so a modest hardware is enough, and LSRec can be used on other platforms based for example, on Linux. In fact, LSRec has been run on Ubuntu 18.04 and a single LSL stream of 128 float channels and irregular sampling rate. Data were recorded correctly with no loss.

### A. SYNCHRONIZATION OF TRIALS

In addition to proper data synchronization, LSRec allows different experimental tests/stages to be linked with their corresponding input data. This task can be accomplished by three explicit sync methods in which markers are saved with data (synchronization threads), and another where markers are recorded as a separate input data stream. The explicit techniques facilitate data segmentation, reducing the probability of errors in the segmentation process, with a deviation of around $\pm 90\mu$s in more than 90% of cases for one-channel streams. This deviation is about $130\mu$s in the worst case (Table 3, sampling rate of 512 Hz: $130\mu\text{s} = 90\mu\text{s} - (-7\mu\text{s} - 33\mu\text{s}) = (90\mu\text{s} - (mean_{512Hz} - std_{512Hz}))$. This way, a sync marker exhibits a deviation of 1 sample with respect to the correct location for a sampling rate of 7.6 kHz, and this may be aggravated in higher sampling rates.

Regarding sync markers, a higher data rate affects LSRec's performance, whatever the selected sync method (LSL streams or socket), because of a reduction of the resources available in the computer. Likewise, the tests' rate was set to 10 messages per second or 100 ms between messages. We consider that this value is high, since a greater rate is uncommon in studies based on living beings.

### B. SWOT ANALYSIS

All systems have positive and negative aspects that need to be mentioned. Strengths-Weaknesses-Oportunities-Threats (SWOT) analysis evaluates the strengths and weaknesses of a software, highlighting its opportunities and threats [14]. Table 5 contains the elements of a SWOT analysis. The main weakness and threat of LSRec is that it is based on a unique external device library, so any variations in that library may stop the system from working. However, the multi-platform and open-source code and its modular design structure enables researchers and programmers to use it in different operating systems and develop new capabilities. Moreover, linking the data segments with an experimental phase is facilitated because the sync methods, and start/stop recording process can be automated using special sync messages.

### C. USABILITY TEST

The analysis of the TAM3 survey shows that subjects intended to use LSRec because it was perceived as useful, although the perceived ease of use did not influence this decision. This may be because LSRec's settings were not detailed in the task guide, and subjects had to work out how to use it. The behavioral intention statements indicate that users intended to use the system (averages: BI1= 6.17, BI2= 6.17, and BI1= 6.50). The items in the BI category exhibited significant dependency on the others. The relationships between BI1 with PU1 ($\rho$= 0.87, $p < 0.05$), and BI1, BI2, and BI3 with PU4 ($\rho >$ 0.92, $p < 0.05$) support the intention to adopt the system because it was seen as useful and with the potential to improve working capacity. Likewise, its pleasantness of use (BI1, BI2, and BI3 with ENJ2; $\rho >$ 0.94, $p < 0.05$), a favorable working environment in which to use it (BI2 and BI3 with SN1 and SN3; $\rho \simeq$ 0.86 and $\rho >$ 0.93, $p < 0.05$ ), and a correct fit of the system in the user's activities (REL1 with BI1 and BI3, and REL2 with BI2 and BI3; $\rho >$ 0.86, $p < 0.05$) all reinforce this idea.

With respect to the perceived usefulness, the pleasantness of use of LSRec directly affected the performance, effectiveness and usefulness for the users (ENJ2 with PU1, PU3, and PU1; $\rho >$ 0.84, $p < 0.05$). At the same time, the social environment, the fit of the application with the user's tasks, and his/her ability to communicate the benefits of the system led users to find the system useful (PU4 with SN1, SN3, REL1, REL3, RES2, and RES3; $\rho >$ 0.88, $p < 0.05$).

Focusing on the perceived ease of use (averages: PEOU1= 6.00, PEOU2= 6.83, PEOU3= 6.33, PEOU4= 6.17), LSRec was rated as easy to use because users were able to perform tasks without any supervision (CSE1 with PEOU1-3; $\rho >$ 0.84, $p < 0.05$), they had control over it and realized that it was compatible with other systems (PEC1 with PEOU1-4; PEC2 with PEOU1, PEOU3-4; PEC4 with PEOU1-2; $\rho >$ 0.83, $p < 0.05$), and the quality of output data was high (OUT1 with PEOU1-3; $\rho >$ 0.86, $p < 0.05$).

## VI. CONCLUSION

LSRec is an easy-to-use, open-source, multi-platform recording system that can save data from several devices and experimental phase markers at the same time with different sampling rates. LSRec was run on Windows 10, two Linux distributions (Ubuntu 18.04 and Ubuntu 20.04), and Mac OS X is pending evaluation. Unlike other systems, adding new data devices is easy and it is not necessary to modify LSRec's code. In addition, we implemented different methods to mark the different experimental stages to record data, to ease data segmentation. One of these was based on socket messages, allowing the compatibility with existing software that send markers using TCP/UDP messages. LSRec allows sync markers to come from more than one source. This gives greater flexibility to the system, compared to others where synchronization mechanics are either missing or are limited to just one source. On the other hand, LSRec implements its own output format (Appendix C) which enables data encryption and reduces the RAM memory issue. The encode process can be performed by several parallel threads, leading to a drastic reduction in time. It also allows users to plot input signals, add extra information to each stream, and it gives a general description about recorded data, and the data recovery process to overcome system problems, for example, system shutdown. It also supplies a start/stop command based on sync markers, thereby improving the control of experimental tests.

This system will keep on improving with new options being made available, such as new output data files (XDF, Matlab, etc.), and a user manual based on user annotations.

## APPENDIX A
## OUTPUT DATA STREAM

LSRec has an unlimited list of supported devices. Data can be recorded from different devices using official supported drivers or the community's contributions, like, Emotiv, Muse, G.tec g.USBAmp, Nintendo Wii control, mouse and keyboard, Kinect, etc. Furthermroe, unlike other systems such as BCI2000 or BCI++, new hardware devices can be added to LSRec without modifications. Other software manages communication with the hardware device and sends data through LSL. This way, LSRec only collects data from different streams, while the other software is responsible for setting and running the LSL streams to send data. Creating a new output data stream to send data to LSRec is easy. The following code is a Java example of this process:

```
String name = "myDevice";
// name of my data source
```

```
String value = "value";
// data descriptor

int ch = 8;
// number of channels = 8

int chunk = 32
// chunk size

double sr = 100;
// sampling rate = 100~Hz

int dataType = LSL.ChannelFormat.float32;
// output data of~32-bit float

String ID = "myuid123456"; // device ID

LSL.StreamInfo info =
new LSL.StreamInfo( name, value, ch
                    , sr, dataType, ID );
// Stream information

LSL.StreamOutlet outlet =
new LSL.StreamOutlet( info );
// To make streaming data

float[] outputData= new float[ch *chunk];
// output data array

// check if a receiver exists
if( outlet.have_consumers() )
{
fillInOutputDataArray( outputData );

outlet.push_chuck( outputData );
// Send data
}
```

## APPENDIX B
## OUTPUT FORMAT

LSRec is a free and open-source software which can be customized by adding new features. The output data format is a good example. A new output format file may be added by creating a new class that implements the interface *IOutputDataFileWriter* which defines five functions:

```
/**
* To add metadata information
* @param id, text: metadata identify
*        and the information string.
*/
public void addMetadata( String id, String text )
throws Exception;

/**
* To write data to output file.
* @param data: data to save.
*        Class with information
*        and output data array.
* @Return: true if data was saved.
*          Otherwise, false.
*/
public boolean saveData( DataBlock data )
throws Exception;

/**
* Return: the file path.
*/
public String getFileName();

/**
* To check if the encoder is finished.
*/
public boolean finished();

/**
* Finish the encoding process and
* close the output file
*/
public void close() throws Exception;
```

LSRec splits data into blocks to avoid memory problems. When a buffer of 10MiB is full, the function *saveData(...)* is called. LSRec defines a template for this interface, through a non-blocking function *saveData(...)* that allows the data segmentation in 10MiB blocks in parallel with the conversion process. Two output formats are supported: HDF5 [15] and CLIS, owner output file format described in the Appendix C.

## APPENDIX C
## CLIS FORMAT

LSRec uses CLIS format version 2.1 to save data into an output file that contains two parts (Figure 6): header (text in UTF-8 format) and data block (binary values).

### A. CLIS v2 HEADER

The header contains the information necessary to restore original data. It is split into 3 parts (Figure 6, on the right): restoration fields, data information data information (DI), and padding.

#### 1) RESTORATION FIELDS

Block of text lines ending with the special character "new line" (\n), containing information to restore original data. In the first line, each field is split by a semicolon. The first three identify the version, the compression technique, and size of header in bytes. The last one indicates whether restoration-field extension is available (true/false), and the previous one indicates the length (in byte) of DI in the CLIS header (a zero value means the DI block is not present). The other middle fields in the first text line, (DBFs), are associated

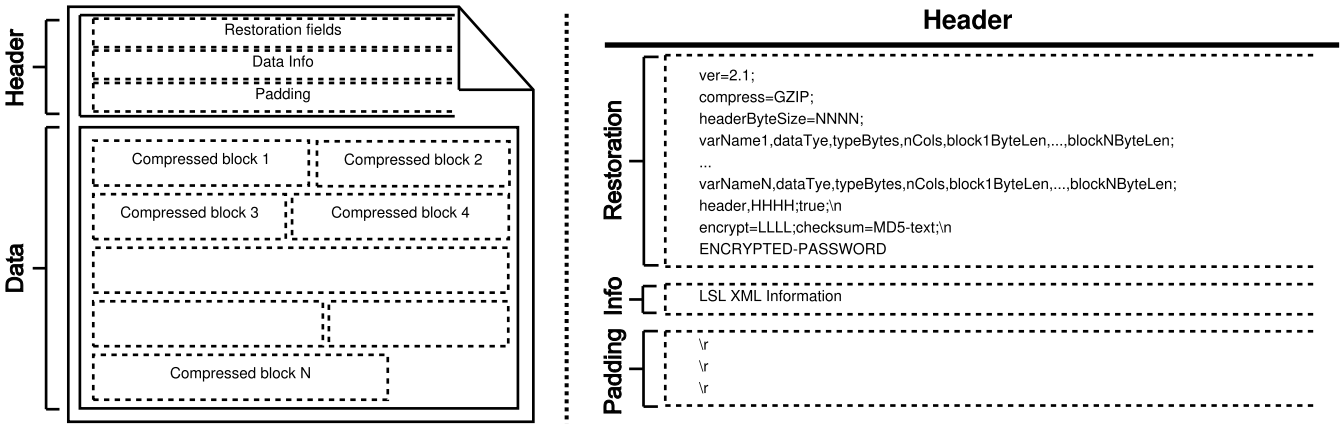


**FIGURE 6.** CLIS v2 file format. On the left, the general scheme, and on the right, the header structure where restoration and info fields are a single text line respectively.

to data blocks. In these, different subfields are separated by a comma with the following meaning (in order):

- *varName:* name of the data stream.
- *dataType:* three exclusive options: int (integer), float, and char (character).
- *typeBytes:* number of bytes of data type. For example, a double value and a 32-bit integer are identified as "*float,8*", and "*int,4*" respectively.
- *nCols:* data stream is structured as an $N \times M$ matrix. This field identifies the number of columns of the matrix.
- *block1ByteLen,...,blockNByteLen:* number of bytes of each input 10MiB-data segment after compression and encryption in order.

The order in which each data-block field (DBF) appears indicates their order in the file.

The last field of the first text line of restoration information shows whether the extension is available. A value equal to *true* indicates complementary information to restore data blocks is contained in the next text line. Version 2.1 of CLIS format is compounded by 2 fields separated by a semicolon. The first one indicates the length (in bytes) of the saved symmetric-encryption key after the next $\backslash n$. This is encrypted, so it is advisable to check that the decryption key is the same (in the Figure 6, the encryption key is indicated by the string *ENCRYPTED-PASSWORD*). The encrypt process is based on Java's implementation identified as *AES/CBC/PKCS5Padding*. The second field contains an MD5-checksum code of encrypted-compressed data. The steps to generate it are: firstly data block (one by one), and then, CLIS header without the checksum code and padding (in the Figure 6, the checksum code is indicated by the string *MD5-text*).

#### 2) DATA INFORMATION

This is an optional block that includes comments about data. In LSRec, this block contains LSL streaming information in XML format. This text is encrypted if the user sets an encrypt key.

#### 3) PADDING

The number of final data blocks and the length of each one are unknown at the beginning of the compression and encryption processes. This means that the number of characters in the *restoration fields* is unknown initially. The aim of the padding is to reserve enough memory space to insert the two previous header blocks at the beginning of the output file without overwriting the compressed data blocks. The header byte size is estimated based on uncompressed data for each 10MiB segment, and the special character "carriage return" ($\backslash r$) is inserted until the estimation is reached.

### B. CLIS v2 DATA BLOCK

The data block consists of a sequence of compressed 10MiB-data segments, which is described in the Section II-E. LSRec currently supports two compression techniques: GZIP (default) and BZIP2. Each compressed block is encrypted if the user sets an encrypt key.

The process to convert the temporary input binary file data to the output data file is split into 3 parts: 1) insert the padding to reserve the header, 2) compress, encrypt and save the input binary data file, and 3) write the header file at the beginning.

In turn, importing CLIS data in Matlab and Python can be done in the LSRec's URL.

## APPENDIX D
## USED TAM3 STATEMENTS

The employed questions of TAM3 survey are showed in Tables 6 and 7

## APPENDIX E
## USER TASK GUIDE

LSLRecorder is a Java application based on Lab Streaming Layer. It records digital signals for offline data analysis. This test assesses the usefulness of LSLRecorder for signal processing research.

1) Start the test and fill in the next questions.

**TABLE 6.** TAM3 survey (part 1). Scoring: from 1 (disagree) up to 7 (agree).

| | TAM3 statements |
|---|---|
| PU | PU1 Using the system improves my performance in my job. |
| | PU2 Using the system in my job increases my productivity. |
| | PU3 Using the system enhances my effectiveness in my job. |
| | PU4 I find the system to be useful in my job. |
| PEOU | PEOU1 My interaction with the system is clear and understandable. |
| | PEOU2 Interacting with the system does not require a lot of my mental effort. |
| | PEOU3 I find the system to be easy to use. |
| | PEOU4 I find it easy to get the system to do what I want it to do. |
| CSE | I could complete the exercise using the software . . . |
| | CSE1 . . . if there was no one around to tell me what to do as I go. |
| | CSE3 . . . if someone showed me how to do it first. |
| PEC | PEC1 I have control over using the system. |
| | PEC2 I have the resources necessary to use the system. |
| | PEC3 Given the resources, opportunities and knowledge it takes to use the system, it would be easy for me to use the system. |
| | PEC4 The system is not compatible with other systems I use. |
| CPLAY | The following questions ask you how you would characterize yourself when you use computers: |
| | CPLAY1 . . . spontaneous |
| | CPLAY2 . . . creative |
| | CPLAY3 . . . playful |
| | CPLAY4 . . . unoriginal |
| ENJ | ENJ1 I find using the system to be enjoyable. |
| | ENJ2 The actual process of using the system is pleasant. |
| | ENJ3 I have fun using the system. |
| OU | OU1 No specifit items were used. It was measured as a ratio of time spent by the subject to the spent by an expert on the same set of tasks. |

**TABLE 7.** TAM3 survey (part 2). Scoring: from 1 (disagree) up to 7 (agree).

| | TAM3 statements |
|---|---|
| SN | SN1 People who are important in my professional environment think that I should use the system. |
| | SN2 People who are important to me think that I should use the system. |
| | SN3 The senior management of this business has been helpful in the use of the system. |
| | SN4 In general, the organization has supported the use of the system. |
| VOL | VOL1 My use of the system is voluntary. |
| | VOL2 My supervisor does not require me to use the system. |
| | VOL3 Although it might be helpful, using the system is certainly not compulsory in my job. |
| REL | REL1 In my job, usage of the system is useful. |
| | REL2 In my job, usage of the system is relevant. |
| | REL3 The use of the system is pertinent to my various job-related tasks. |
| OUT | OUT1 The quality of the output I get from the system is high. |
| | OUT2 I have no problem with the quality of the system's output. |
| | OUT3 I rate the results from the system to be excellent. |
| RES | RES1 I have no difficulty telling others about the results of using the system. |
| | RES2 I believe I could communicate to others the consequences of using the system. |
| | RES3 The results of using the system are apparent to me. |
| | RES4 I would have difficulty explaining why using the system may or may not be beneficial. |
| BI | BI1 I intend to use it. |
| | BI2 I predict that I would use it. |
| | BI3 I plan to use the system in the future. |

a) How many streams are there? What is the name, data type, number of channels, and sampling rate of each one? Did you need help?
b) Plot the streams and indicate what the input stream types are (sinusoidal, sawtooth, or random) in each one? Did you need help?
c) Add extra information to each stream with the next format: type=*class*, where *class* is sinusoidal, sawtooth, or random. Example: type=linear. Did you need help?

2) Execute the following activity, and refresh the input streams. Did you need help?
   a) Select all streams. Did you need help?
   b) Rename the output data file: "dataXY1.clis". The symbol X means your name and symbol Y is your family name. Did you need help?
   c) Start recording and wait for the capture to end. What was the start time? How did you know? What was the end time? How did you know that the recording was over? Did you need help?
3) Execute the following activity, refresh the input streams and select all streams.
   a) Rename the output data file: "dataXY2.clis". The symbol X means your name and symbol Y is your family name.
   b) Select Socket as sync method. Did you need help?
   c) Set UDP as network protocol, 127.0.0.2 as IP address, and port number 12345. Did you need help?
   d) Insert the next socket's input messages: *udp1*, *udp2*, *udp3*, and *udp4*. What are the sync markers of each message. Did you need help?
   e) Delete *upd3*. Refill in the sync markers.
   f) Start recording and wait for the capture to end. How many input messages were received? Why did the recorder finish? What was the input message pattern? Did you need help?
4) Refresh the input streams.
   a) Rename the output data file: "dataXY3.clis". The symbol X means your name and symbol Y is your family name.
   b) Activate *special input messages* (start/stop recording). Did you need help?
   c) Select Socket as sync method and set UDP as network protocol, 127.0.0.3 as IP address, and 23456 as port.
   d) Preserve previous socket messages table.
   e) Clear the input message log. Did you need help?
   f) Start recording, execute 03-captura3.vbs, and wait for the capture to end. How many input

messages were received? Why did the recorder finish? What was the input message pattern?

5) Execute the following activity, and refresh the input streams.
   a) Rename the output data file: ''dataXY4.clis''. The symbol X means your name and symbol Y is your family name.
   b) Select all streams. A new stream is detected. Its name is ''flujo_6''. Its chunk size is 10 samples by channel. Data are interleaved, that is, for 3 channels, the 1st sample is from the 1st channel, the 2nd sample is from the 2nd channel, and the 3rd one is from the 3rd channel, and so on. Set the chunk size to 10 and active the interleaved. Did you need help?
   c) Plot the stream ''flujo_6''. Three sine signal must be plotted, on the contrary, check the settings.
   d) Select Socket as sync method and set TCP as network protocol, 127.0.0.4 as IP address, and 34567 as port.
   e) Deactivate *special input messages*.
   f) Clear the input message log, remove all socket messages and insert the next ones: *tcp1*, *tcp2*, and *tcp3*.
   g) Start recording, and wait for the capture to end. How many input messages were received? What was the input message pattern?

6) Execute the following activity, and refresh the input streams.
   a) Rename the output data file: ''dataXY5.clis''. The symbol X means your name and symbol Y is your family name.
   b) Select Lab-Streaming Layer as sync method. Did you need help?
   c) Check that *special input messages* is not selected.
   d) Select all streams.
   e) Set as synchronization streams those that can be selected as a sync method. What were the stream's names? What was the reason for a stream being set as sync input thread? Did you need help?
   f) Clear the input message log, start recording, and wait for the capture to end. How many input messages were received? Why did the recorder finish? What was the input message pattern?

7) Execute the following activity, and refresh the input streams.
   a) Rename the output data file: ''dataXY6.clis''. The symbol X means your name and symbol Y is your family name.
   b) Select Lab-Streaming Layer as sync method.
   c) Activate *special input messages*.
   d) A new stream is available to set as sync thread. Deselect the previous sync stream and select this new one as sync stream. Did you need help?
   e) Clear the input message log, start recording, and wait for the capture to end. How many input messages were received? Why did the recorder finish? What was the input message pattern?

## ACKNOWLEDGMENT

The authors would like to thank the volunteers from the Departamento de Tecnología Electrónica of Universidad de Sevilla (Spain) for assessing LSRec; Patrick Partridge who thoroughly revised the manuscript; and the anonymous reviewers for their useful suggestions.

## CONFLICT OF INTEREST

The authors declare no conflict of interest.

## REFERENCES

[1] P. Ammann and J. Offutt, *Introduction to Software Testing*. New York, NY, USA: Cambridge Univ. Press, 2012.
[2] R. Barea, L. Boquete, M. Mazo, and E. Lopez, ''System for assisted mobility using eye movements based on electrooculography,'' *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 10, no. 4, pp. 209–218, Dec. 2002.
[3] W. Boucsein, D. C. Fowles, S. Grimnes, G. Ben-Shakhar, W. T. Roth, M. E. Dawson, and D. L. Filion, ''Publication recommendations for electrodermal measurements,'' *Psychophysiology*, vol. 49, no. 8, pp. 1017–1034, Aug. 2012.
[4] P. Bourque and R. E. Fairley, Eds., *SWEBOK: Guide to the Software Engineering Body of Knowledge*, IEEE Computer Society, Los Alamitos, CA, USA, version 3.0 edition, 2014.
[5] C. Brunner, G. Andreoni, L. Bianchi, B. Blankertz, C. Breitwieser, S. Kanoh, A. Christian Kothe, A. Lécuyer, S. Makeig, J. Mellinger, P. Perego, Y. Renard, G. Schalk, I. Putu Susila, B. Venthur, and R. Gernot Müller-Putz, *BCI Softw. Platforms*, Berlin, Germany: Springer, 2013, pp. 303–331.
[6] J. A. Castro-Garcia, A. J. Molina-Cantero, M. Merino-Monge, and I. M. Gomez-Gonzalez, ''An open-source hardware acquisition platform for physiological measurements,'' *IEEE Sensors J.*, vol. 19, no. 23, pp. 11526–11534, Dec. 2019.
[7] J. A. Castro-García, ''Adquisición y procesamiento de señales psiológicas y sus aplicaciones,'' Ph.D. dissertation, Dept. Electron. Technol., Univ. Sevilla, Seville, Spain, Oct. 2019.
[8] G. Chanel, C. Rebetez, M. Bétrancourt, and T. Pun, ''Emotion assessment from physiological signals for adaptation of game difficulty,'' *IEEE Trans. Syst., Man, Cybern., A, Syst. Humans*, vol. 41, no. 6, pp. 1052–1063, Nov. 2011.
[9] A. Collette, *Python and HDF5: Unlocking Scientific Data*. Sebastopol, CA, USA: O'Reilly Media, 2013.
[10] B. R. Cox, A. C. Stolte, K. H. Stokoe, and L. M. Wotherspoon, ''A direct-push crosshole (DPCH) test method for the *in situ* evaluation of high-resolution P- and S-Wave velocities,'' *Geotechnical Test. J.*, vol. 42, no. 5, Sep. 2019, Art. no. 20170382.
[11] S. J. Dumas and C. J. Redish, *A Practical Guide to Usability Testing*, 1st ed. Exeter, U.K.: Intellect Books, 1999.
[12] M. Duvinage, T. Castermans, M. Petieau, T. Hoellinger, G. Cheron, and T. Dutoit, ''Performance of the emotiv epoc headset for P300-based applications,'' *Biomed. Eng. OnLine*, vol. 12, no. 1, p. 56, 2013.
[13] S. H. Fairclough and K. Gilleade, Eds., *Advances in Physiological Computing* (Human-Computer Interaction Series). London, U.K.: Springer, 2014.
[14] L. G. Fine, ''The SWOT analysis: Using your strength to overcome weaknesses, using opportunities to overcome threats,'' CreateSpace Independent Publishing Platform, Scotts Valley, CA, USA, Tech. Rep., 2009.
[15] M. Folk, G. Heber, Q. Koziol, E. Pourmal, and D. Robinson, ''An overview of the HDF5 technology suite and its applications,'' in *Proc. EDBT/ICDT Workshop Array Databases (AD)*, 2011, pp. 36–47.
[16] A. Gibaldi, M. Vanegas, J. Peter Bex, and G. Maiello, ''Evaluation of the Tobii EyeX Eye tracking controller and MATLAB toolkit for research,'' *Behav. Res. Methods*, vol. 49, pp. 923–946, Jul. 2017.
[17] K. Gramann, P. Daniel Ferris, J. Gwin, and S. Makeig, *Imaging Natural Cognition in Action*. 2014.
[18] N. W. Group, P. Deutsch, and A. Enterprises, *GZIP file Format Specification Version 4.3, Distribution*, RFC, document 1952, 1996.

[19] A. Kappas, "Smile when you read this, whether you like it or not: Conceptual challenges to affect detection," *IEEE Trans. Affect. Comput.*, vol. 1, no. 1, pp. 38–41, Jan. 2010.
[20] F. Kong and Y. Wang, "Multimodal interface interaction design model based on dynamic augmented reality," *Multimedia Tools Appl.*, vol. 78, pp. 4623–4653, Jul. 2018.
[21] S. Kundu and S. Ari, "P300 detection with brain–computer interface application using PCA and ensemble of weighted SVMs," *IETE J. Res.*, vol. 64, no. 3, pp. 406–414, 2018.
[22] J. Lindgren and A. Lecuyer, "OpenViBE and other BCI software platforms," in *Brain-Computer Interfaces 2: Technology and Applications*, M. Clerc, L. Bougrain, and F. Lotte, Eds. London, U.K.: Wiley, 2016.
[23] Y. Liu, X. Jiang, T. Cao, F. Wan, P. U. Mak, P.-I. Mak, and M. I. Vai, "Implementation of SSVEP based BCI with emotiv EPOC," in *Proc. IEEE Int. Conf. Virtual Environ. Human-Computer Interfaces Meas. Syst. (VECIMS)*, Jul. 2012, pp. 34–37.
[24] M. Merino, I. Gomez, and A. J. Molina, "EEG feature variations under stress situations," in *Proc. 37th Annu. Int. Conf. IEEE Eng. Med. Biol. Soc. (EMBC)*, Aug. 2015, pp. 6700–6703.
[25] M. Merino, I. Gómez, O. Rivera, and J. A. Molina, "Customizable software interface for monitoring applications," in *Computers Helping People with Special Needs. ICCHP* (Lecture Notes in Computer Science), vol. 6179, K. Miesenberger, J. Klaus, W. Zagler, and A. Karshmer, Eds. Berlin, Germany: Springer, 2010.
[26] A. Molina, J. Guerrero, I. Gómez, and M. Merino, "A new multisensor software architecture for movement detection: Preliminary study with people with cerebral palsy," *Int. J. Hum.-Comput. Stud.*, vol. 97, pp. 45–57, Jan. 2017.
[27] A. Molina-Cantero, J. Guerrero-Cubero, I. Gómez-González, M. Merino-Monge, and J. Silva-Silva, "Characterizing computer access using a one-channel EEG wireless sensor," *Sensors*, vol. 17, no. 7, p. 1525, Jun. 2017.
[28] G. C. Moore and I. Benbasat, "Development of an instrument to measure the perceptions of adopting an information technology innovation," *Inf. Syst. Res.*, vol. 2, no. 3, pp. 192–222, Sep. 1991.
[29] L. Muller, A. Bernin, S. Ghose, W. Gozdzielewski, Q. Wang, C. Grecos, K. von Luck, and F. Vogt, "Physiological data analysis for an emotional provoking exergame," in *Proc. IEEE Symp. Ser. Comput. Intell. (SSCI)*, Dec. 2016, pp. 1–8.
[30] R. Näätänen and T. Picton, "The n1 wave of the human electric and magnetic response to sound: A review and an analysis of the component structure," *Psychophysiology*, vol. 24, no. 4, pp. 375–425, Jul. 1987.
[31] NASA. (2008). *XDF: TheExtensible Data Format Based on XML Concepts*. Accessed: Mar. 11, 2019. [Online]. Available: https://nssdc.gsfc.nasa.gov/nssdc_news/june01/xdf.html
[32] M. Noback, *Principles of Package Design: Creating Reusable Software Components*. New York, NY, USA: Apress, 2018.
[33] P. Perego, L. Maggi, S. Parini, and G. Andreoni, "BCI++: A new framework for brain computer interface application," in *Proc. 18th Int. Conf. Softw. Eng. Data Eng. (SEDE)*, 2009, pp. 1–6.
[34] P. Pérez, G. Huertas, A. Maldonado-Jacobi, M. Martín, J. A. Serrano, A. Olmo, P. Daza, and A. Yúfera, "Sensing cell-culture assays with low-cost circuitry," *Sci. Rep.*, vol. 8, no. 1, p. 8841, Dec. 2018.
[35] T. Pfister and P. Robinson, "Real-time recognition of affective states from nonverbal features of speech and its application for public speaking skill analysis," *IEEE Trans. Affect. Comput.*, vol. 2, no. 2, pp. 66–78, Apr. 2011.
[36] G. Pfurtscheller, C. Brunner, A. Schlögl, and F. L. D. Silva, "Mu rhythm (de) synchronization and EEG single-trial classification of different motor imagery tasks," *NeuroImage*, vol. 31, no. 1, pp. 153–159, May 2006.
[37] R. Quesada-Tabares, J. A. Molina-Cantero, I. Gómez-González, M. Merino-Monge, J. A. Castro-García, and R. Cabrera-Cabrera, "Emotions detection based on a single-electrode EEG device," in *Proc. 4th Int. Conf. Physiol. Comput. Syst.* Madrid, Spain: SciTePress, 2017, pp. 89–95.
[38] P. M. R. Reis, F. Hebenstreit, F. Gabsteiger, V. von Tscharner, and M. Lochmann, "Methodological aspects of EEG and body dynamics measurements during motion," *Frontiers Human Neurosci.*, vol. 8, p. 156, Mar. 2014.
[39] 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, Jun. 2004.
[40] C. Setz, B. Arnrich, J. Schumm, R. La Marca, G. Troster, and U. Ehlert, "Discriminating stress from cognitive load using a wearable EDA device," *IEEE Trans. Inf. Technol. Biomed.*, vol. 14, no. 2, pp. 410–417, Mar. 2010.
[41] R. A. Stevenson and T. W. James, "Affective auditory stimuli: Characterization of the international affective digitized sounds (IADS) by discrete emotional categories," *Behav. Res. Methods*, vol. 40, no. 1, pp. 315–321, Feb. 2008.
[42] V. Venkatesh, "Determinants of perceived ease of use: Integrating perceived behavioral control, computer anxiety and enjoyment into the technology acceptance model," *Inf. Syst. Res.*, vol. 11, no. 4, pp. 342–365, 2000.
[43] V. Venkatesh and H. Bala, "Technology acceptance model 3 and a research agenda on interventions," *Decis. Sci.*, vol. 39, no. 2, pp. 273–315, May 2008.
[44] J. Webster and J. J. Martocchio, "Microcomputer playfulness: Development of a measure with workplace implications," *MIS Quart.*, vol. 16, no. 2, p. 201, Jun. 1992.
[45] L. Yang, L. Zhang, H. Dong, A. Alelaiwi, and A. E. Saddik, "Evaluating and improving the depth accuracy of kinect for windows v2," *IEEE Sensors J.*, vol. 15, no. 8, pp. 4275–4285, Aug. 2015.

**MANUEL MERINO-MONGE** was born in Seville, Spain, in 1983. He received the master's degree in computer engineering from the University of Seville, in 2010, and the Ph.D. degree from the TAIS Research Group (Technologies for Care, Inclusion and Health), University of Seville, in 2015. His research interests include biomedical signal processing, affective computing, human–computer interface, augmentative and alternative communication, and assistive technology.

**ALBERTO J. MOLINA-CANTERO** was born in Lucena, Córdoba, in 1967. He received the B.S. degree in physics with intensification in electronics, in Seville, in 1990, and the Ph.D. degree from the University of Seville, in 2010. He has been working as a Professor with the University of Seville, since 1990. He is the author of five books, more than 30 research articles, holds three inventions, and took part in more than ten projects. His research interests include signal processing, sensors, and the development of new devices or techniques for helping people to access a computer.

**JUAN A. CASTRO-GARCÍA** (Graduate Student Member, IEEE) was born in Lora del Río, Seville, in 1986. He received the master's degree in computers and networks engineering from the Universidad de Sevilla, in 2011, and the Ph.D. degree from the TAIS Research Group (Technologies for Care, Inclusion and Health), University of Seville, in 2019. His research interests include signal processing, sensors, and human–computer interfaces.

**ISABEL M. GÓMEZ-GONZÁLEZ** (Senior Member, IEEE) has been a Senior Researcher and a full-time Ph.D. Lecturer with the Electronic Technology Department, Universidad de Sevilla, since 1990. Her teaching is centered on digital electronic and biomedical technology disciplines in bachelor's and master's degrees of computer science. She has directed numerous final career projects about the design and implementation of serious games and toys adaptation in order to improve the cognitive and motor skills of adults and children with cerebral palsy. She is also the Head of the TAIS Research Group (Technologies for Care, Inclusion and Health), where she has been managing several Ph.D. thesis and national research projects related to human–computer interfaces, biosignal processing, and ambient assisted living.