A Procedure for Measuring Latencies in Brain–Computer Interfaces

J. Adam Wilson*, *Member, IEEE*, Jürgen Mellinger, Gerwin Schalk, *Member, IEEE*, and Justin Williams, *Member, IEEE*

***Abstract*—Brain–computer interface (BCI) systems must process neural signals with consistent timing in order to support adequate system performance. Thus, it is important to have the capability to determine whether a particular BCI configuration (i.e., hardware and software) provides adequate timing performance for a particular experiment. This report presents a method of measuring and quantifying different aspects of system timing in several typical BCI experiments across a range of settings, and presents comprehensive measures of expected overall system latency for each experimental configuration.**

***Index Terms*—Brain–computer interface (BCI), BCI2000, real time, timing.**

I. Introduction

FOR DECADES, experiments in cognitive research have relied on computers for data collection and subject interaction. These experiments often depend on accurate timing of events, e.g., to present stimuli or to measure a subject's reaction time [1], [2]. However, the most prevalent operating system on computers today, Microsoft Windows, is not ideally suited to support accurate timing [3]–[6]. Other operating systems, such as Linux or Mac OS X, may provide more accurate control over system timing [5], but they often lack the required software and drivers for amplification and acquisition systems. As a result, most experiments in cognitive research currently rely on external hardware for precise event timing that is independent of the operating system and computer configuration. A typical configuration may consist of software that presents stimuli on a video monitor time-locked to the refresh cycle of the screen. Timing of stimulus presentation, as well as other external input, such as key presses, could be encoded as digital signals that can be digitized along with neural signals. While such technical approaches provide highly accurate timing, they are complicated to set up and limited in the experimental complexity that they can support. For example, an experiment that uses a portable laptop system may be unable to interface with the external timer, and therefore, the stimulus timing could not be verified on this system.

The study of brain–computer interfaces (BCIs) [7] is an emerging area of research that shares many principles with cognitive research. Similar to cognitive research, a subject is presented with stimuli and the subject's brain signal responses to these stimuli are detected using different sensor methods, such as electroencephalography (EEG) [8], electrocorticography (ECoG) [9], [10], or single-neuron recordings [11]. However, an additional requirement of BCIs is that the stimuli are continuously changing, based on measurements derived from current and previous neural signals, such that the user can learn to control the output using only voluntarily modulated brain activity. Thus, a BCI system must control or detect event timing, such as when the stimulus is updated and the user responds. Additionally, it must also ensure that the system is capable of performing online signal processing and classification, so that task-related changes in the brain signal are properly detected and results are submitted to the output device (e.g., a video screen or other external device, such as a robotic arm) with minimal delay. While such a system could, in principle, be implemented using the technical approaches described earlier (i.e., hardware-based timing/triggers, or real-time capable operating systems), its implementation would be complex, and hence, time-consuming. Furthermore, it would usually also be very specific to the particular characteristics of individual experiments. This is a problem, because the current early stage of development in BCI research implies that many factors of BCI systems, such as the utilized brain signals, signal processing algorithms, or feedback modalities, need to be evaluated to optimize BCI performance [12].

Fortunately, the demands in timing precision in BCI research are typically somewhat more relaxed compared to other experiments in cognitive research. For example, accurate measurement of the amplitude of the mu rhythm, i.e., an oscillation around 8–12 Hz over sensorimotor cortex that is modulated by movements, will not be substantially affected by small delays (e.g., 10 ms) in either the amplitude measurement, stimulus presentation, or the resulting feedback update. As an example for

Manuscript received April 8, 2009; revised September 9, 2009 and January 12, 2010; accepted February 16, 2010. Date of publication April 15, 2010; date of current version June 16, 2010. The work of J. Williams was supported by the National Institutes of Health under Clinical Neuroengineering Training Program 1-T90-DK070079-01. The work of J. Williams and National Center for Research Resources was supported by the National Institutes of Health under Institutional Clinical and Translational Science Award 1KL2RR025012-01. The work of G. Schalk and National Institute of Biomedical Imaging and Bioengineering (NIBIB) was supported by the National Institutes of Health under Grant R01-EB006356. The work of J. A. Wilson, G. Schalk, and NIBIB was supported by the National Institutes of Health under Grant EB00856. The work of G. Schalk was supported by the U.S. Army Research Office under Grant W911NF-07-1-0415 and Grant W911NF-08-1-0216. *Asterisk indicates corresponding author.*

*J. A. Wilson is with the Department of Neurosurgery, University of Cincinnati, Cincinnati, OH 45219 USA (e-mail: adam.wilson@uc.edu).

J. Mellinger is with the Institute for Medical Psychology and Behavioral Neurobiology, University of Tübingen, Tübingen 72076, Germany.

G. Schalk is with the New York State Department of Health, Wadsworth Center, Albany, NY 12201 USA.

J. Williams is with the Department of Biomedical Engineering, University of Wisconsin-Madison, Madison, WI 53706-1481 USA.

Digital Object Identifier 10.1109/TBME.2010.2047259

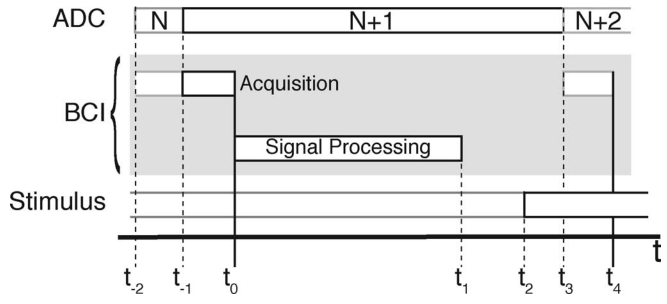

Fig. 1. System timing diagram showing the time line of events in a typical BCI experiment. t_0 is chosen as the time at which the neural data is available to the BCI for processing and marks the onset of a block of data.

the more time-critical nature of traditional psychophysiological experiments, amplitude measurements of auditory brainstem evoked potentials (whose time course is very short), critically depend on a highly accurate relationship between stimulus presentation and brain signals. In summary, the circumstances and requirements of BCI research described earlier suggest that it is both desirable and practical to design a software-based general-purpose BCI system that can be used to implement a range of BCI designs [12]–[15].

Any BCI system implements a closed-loop system that involves the user and an output device. It is typically comprised of an amplification and digitization system that records neural signals, a data acquisition system (i.e., a computer that acquires the neural signals from the amplification and digitization system), a signal processing stage that extracts signal features and translates them into output control signals, and the application output (such as a display on a video monitor, sound through speakers, or other neuroprosthetic device). The effectiveness of a BCI, and the ability of users to learn how to use it, depends on the ability of the system to acquire and process signals, and to present stimuli, in real time, and on providing the user with consistent feedback with low latency and minimal jitter. Because these timing characteristics necessarily vary with different hardware, processing demands, and outputs, a process is required that comprehensively characterizes a particular BCI implementation. However, no previous effort has described such a procedure.

In this paper, we describe a procedure to quantify BCI system latencies at each step in the processing chain. This process allows determination of whether or not the timing characteristics of a particular BCI implementation (i.e., hardware and software) can support the requirements of a particular set of BCI experiments. The procedure is applicable to any BCI system, and is demonstrated for the BCI2000 system [12] and a MATLAB-based BCI implementation. The analysis program and instructions for using any BCI system are included with the BCI2000 distribution, which is freely available on `www.bci2000.org`.

II. Methods

The time line of events in a typical online, closed-loop BCI experiment is shown in Fig. 1. Neural data are acquired in sample blocks; the number of samples in a block is dependent on the sampling rate, e.g., a 30-ms block of data sampled at 1000 Hz contains 30 samples per channel. In Fig. 1, during the period from t_{-2} to t_{-1}, the BCI system waits for the AD converter (ADC) buffer to fill with the required number of samples for block N. At t_{-1}, the ADC buffer is filled with data, and the computer begins reading the data and storing it in memory on the computer for processing. At the same time during the data transfer, the ADC is already recording samples in the next block of data block $N+1$. The period from t_{-1} to t_0 is the time required to transmit all of the data from the ADC to the computer's memory.

At t_0, the BCI system begins processing data to extract relevant signal features and to generate control signals based on the brain activity in block N. Processing continues until t_1, at which point the control signal is ready, and the output command is issued, e.g., to update the monitor, present a sound, or control a device. However, due to latencies in the operating system and the device itself, this stimulus will not be presented immediately. This results in an output delay, which is defined as the time between t_1 (i.e., the time at which the output command is issued) and t_2 (i.e., the time at which the output device actually implements this command). Once the output command is issued in software at t_1, the BCI immediately begins to wait for the next block of data. Block $N+1$ is ready at t_3, and is read into the computer at t_4 (i.e., for block $N+1$, t_3 and t_4 are equivalent to t_{-1} and t_0 relative to block N). For the purposes of this study, the term "real time" is used somewhat loosely, indicating only that the BCI system is able to process an entire block of data and update the output device before the next sample block is ready for processing. However, the degree to which the BCI system can be considered real time is dependent on many factors, including the operating system, computer hardware specifications, and output device hardware. Furthermore, even if the mean overall processing and output time is less than the sample block size, large variability in this timing can significantly affect BCI performance, and must therefore be accounted for.

In summary, each of the BCI processing stages described earlier has an associated latency, which corresponds to the time required to complete a specific task. The duration of these latencies will change depending on the computer hardware and the task configuration. The latencies are defined further in the following section.

A. Latency Definitions

1) ADC Latency: The ADC latency is the delay between the time that the final sample in a sample block is digitized to when the sample block has been acquired by the software and is available to the software for processing. Depending on configuration, this latency may comprise physical signal delay in the amplifier, digitization, transmission from the ADC to the PC, and processing time inside a hardware driver. Because data are transmitted in blocks of one or more of samples, that minimum delay occurs for the last sample of a data block, which unlike its preceding samples in the same block, will spend only a minimum time in hardware and software buffers. Using the times defined in Fig. 1

$$\text{ADC latency} = L_A = t_0 - t_{-1}. \tag{1}$$

For ADCs connected via bandwidth-limited serial interfaces, transmission latency may have a measurable impact on ADC latency. When using universal serial bus (USB) 2.0 or peripheral component interconnect (PCI) card connections, transmission latency may generally be neglected, as illustrated by the following example. If we assume acquisition of 16 channels of 32-bit (4 B) data and transmission of blocks of eight samples, each block corresponds to 512 Bs of data. At USB 2.0 speed (i.e., a maximum transmission rate of 60 MB/s), these data should take approximately 8.5 μs to transfer, which is less than the duration of a single sample. However, if the configuration is changed to 64 channels of 100 ms duration sampled at 4800 Hz, equal to 122 880 Bs, this translates to a transmission time of approximately 2 ms, assuming that transmission starts instantaneously and without interruption. Because this is on the time scale of events during the BCI task, it may be important to account for this latency.

2) Processing Latency: The processing latency is defined as the total time required for the data to be processed, which generally includes extracting task-related information from the neural signals, translating the features into control signals, and finally using these control signals to send an update command to the application or device. From Fig. 1

$$\text{Processing latency} = L_{SP} = t_1 - t_0. \tag{2}$$

The processing latency depends on the algorithmic complexity and on CPU speed. For example, the signal processing algorithm for a cursor movement task might calculate the power spectral density for every block of data, whereas a processing algorithm that extracts an evoked potential might average stimulus-triggered responses on one or more channels. The computational complexity of these two different types of processing may be very different. Thus, the requirements of experiment that is performed will likely dictate the minimum system configuration possible. In either case, it is clear that the system must be capable of processing data faster than the duration of the sample block. Otherwise, the performance of the BCI system will degrade or fail.

3) Output Latency: The output latency is defined as the delay from the time that the output command is issued to the time that the output device implements this command, i.e., the time from calling a software function to present a stimulus on a screen to the time that the stimulus actually appears on the screen. Because it is possible to have any number of output modalities, including video, sound, or mechanical, the output latency depends on the specific modality. From Fig. 1

$$\text{Output latency} = L_O = t_2 - t_1. \tag{3}$$

The output latency can be determined by several contributing factors. For the most common output, a video screen, these factors include the graphics card, the number of monitors used, the resolution, the refresh rate, and even the type of monitor, i.e., CRT or liquid crystal display (LCD).

4) System Latency and System Jitter: The system latency is defined as the minimum time interval between a change in ADC input, and causally related change in the application output. This is the time from t_{-1} to t_2, and is calculated as the sum of the other three latencies

$$\text{System latency} = (t_0 - t_{-1}) + (t_1 - t_0) + (t_2 - t_1)$$
$$\text{System latency} = t_2 - t_{-1}. \tag{4}$$

The system latency jitter is the standard deviation of the system latencies in a given test, and provides a measure of the variability in overall system timing.

5) Block Duration and Block Jitter: The block duration is the time interval between successive blocks of data that have been transmitted to the computer

$$\text{Block duration} = t_4 - t_0. \tag{5}$$

Ideally, the block duration should be identical to the sample block size; however, inconsistencies in operating system timing may interrupt and delay data acquisition, causing the time period between data blocks to be different than the actual block size, introducing a timing jitter. The block jitter is the standard deviation of (5) for all block durations in a single test.

The block duration is the primary indicator of the system's ability to perform online signal processing in a BCI experiment. It is important to realize that the block duration is measured from the perspective of the software, and is not the same as the block size. That is, the block duration will never typically be less than the block size (i.e., the length of a block of data acquired from the ADC), but it *can* be longer than the block size if the system latency is longer than the block size. If the block duration is longer than the block size, this indicates that the time required to process a block of data is longer than the block itself, i.e., the system is still processing block N when block $N + 1$ is ready to be transferred and processed. In this case, the options are to modify the task configuration, use a more powerful system, or optimize processing algorithms to increase performance.

These timing characteristics apply to any BCI system, i.e., all systems that record and process neural data and generate a device command will have some latency between a volitional change in the neural state and corresponding change in the device state, regardless of the source of neural data (e.g., EEG, ECoG, or spikes) and the output device (e.g., a robotic arm, computer cursor, or spelling application). Next, we will describe the requirements for measuring these latencies in any BCI system.

B. Requirements for Measuring Latencies

Accurate measurement of system latencies requires methods for determining the precise timing of the events in Fig. 1. This section describes a series of methods for measuring these latencies without the use of an external event timer, using only the ADC and software time stamps.

1) ADC Latency: To measure the ADC latency, it is necessary to record the times immediately preceding the data read operation (i.e., t_{-1} in Fig. 1) and immediately following the read (i.e., t_0 in Fig. 1). However, t_{-1} is defined as the beginning of data block $N + 1$, corresponding to sample 0 in block $N + 1$, and therefore, does not need an associated time stamp.

If the ADC system contains one or more digital output channels that can be controlled by software, then it is possible to use that digital output channel as an event marker that can be recorded on an input channel. Otherwise, an alternative means of sending an event marker to the amplifier, and that can be controlled from software, is required. This could be accomplished with a separate digital-to-analog convertor (DAC) board (e.g., from National Instruments), or any other controllable output device. To do so, the digital output (whether on the amplifier hardware or an external DAC) must be set high (or pulsed) at t_0 in Fig. 1, when the data transfer is complete. This pulse is then recorded back into the ADC, either as an analog channel or on a separate digital input channel. Since the first sample of the next block ($N + 1$) of data stored in the ADC coincides with the start of the data transfer of block N (i.e., when the last sample has been acquired), the time of the rising edge of the recorded digital channel in block $N + 1$ (measured from the start of the block) corresponds to the *end* of the data transfer for block N.

The time resolution available with this method is dependent on the sampling rate. For example, for a sampling rate of 512 Hz, the time resolution is approximately 1.95 ms, whereas for a sampling rate of 4.8 kHz, it is approximately 0.21 ms. Therefore, a measured ADC latency of 0 ms does not imply that there was an instantaneous data transfer from the ADC to the PC, but rather that the transfer latency was less than the resolution of a single sample at a particular sampling rate.

2) Processing Latency: The processing latency typically constitutes the largest overall latency, and will scale with the amount of data processed. In order to measure the processing latency, one could employ a method similar to that for measuring the ADC latency, i.e., pulsing a digital channel at appropriate events. However, this method may not be optimal for two reasons. First, many ADCs have at most one digital output channel; as described, this digital channel is already being used to measure the ADC latency, and using a single digital channel for more than one event would complicate the analysis. The second and more technically problematic reason is that some BCI systems use a modular system specification in which the data acquisition module runs in an entirely separate program from the processing module. In such cases, the processing module does not have access to ADC functionality to control the digital channels, and therefore, cannot use the digital output as an event timer.

Therefore, our procedure uses software timing functions to record time stamps at the beginning and end of signal processing. In this case, the processing latencies are much higher (possibly tens of milliseconds) than the ADC latencies, and time resolution on the scale of 1 ms is appropriate to measure the processing latency. Furthermore, since our procedure measures the timing of many trials, the mean and standard deviation of the processing latency provide an accurate measure of the processing time.

The specific timing function used will depend on the operating system and the programming language used for the BCI. For Microsoft Windows-based BCIs using C++, the `QueryPerformanceCounter` and `QueryPerformanceFrequency` functions provide accurate timing and were used in this study. On systems with multiple cores, these functions can be buggy if called from multiple threads; therefore, all of the timing for a single application should be handled within a single thread, as is the case with BCI2000.

It is important to note that these values must be saved for offline analysis following the experiment. The time stamps can either be stored directly in the BCI data file along with the EEG data and other event markers, or in a separate log file containing only the time stamps.

3) Output Latency: It is impossible to use software timing to measure the output latency, since a physical change in the output device occurs externally from the software and PC. That is, there is some delay between issuing a command to change the output, and when a measurable change in the output device actually occurs, and this delay cannot be known within the program. Therefore, a stimulus detection system is required to measure outputs and generate an event signal that can be recorded.

The most common BCI outputs are visual (i.e., video on a computer monitor) and auditory (i.e., sound output through a speaker or headphones). In order to detect visual changes, a photodiode can be placed on the monitor to detect display changes related to the experiment; to detect auditory changes, a microphone or direct output from the sound card can be used. These sensors interface with a stimulus detection system, which can be adjusted for the particular intensity levels of each individual stimulus, and should generate a signal when the stimulus is detected. For example, when the optical level surpasses some user-set threshold, the detection system might output a digital pulse. This pulse would be recorded as an analog channel on the ADC, thus providing an accurate event marker for any stimulus with a time resolution equal to the sampling rate. Any such device with this functionality could be purchased (e.g., the g.TRIGbox from g.tec, or the StimTracker from Cedrus), or could be built using a relatively simple circuit.

4) Block Duration and Jitter: Using the methods described earlier, two possible methods exist for measuring the block duration and jitter. The first one uses the software time stamps recorded at the onset of signal processing (i.e., at t_0). The block duration is simply the time difference between the time stamps for consecutive data blocks, i.e., $t_4 - t_0$. Alternatively, the signal recorded from the digital output to determine the ADC latency can be used in a similar manner. The rising edge of this signal also corresponds with t_0, t_4, etc., and therefore, the difference in times between consecutive rising edges will equal the block duration. If the software timer is accurate, then the measured block duration measured using software time stamps should be the same as the hardware method.

5) System Latency: Finally, the overall system latency is determined as a combination of the ADC, processing, and output latencies. Therefore, the system latency measurement will be comprised of a combination of hardware and software event time stamps as described earlier.

C. Hardware

This section describes two specific BCI implementations that we used for the tests presented later in this paper. The first

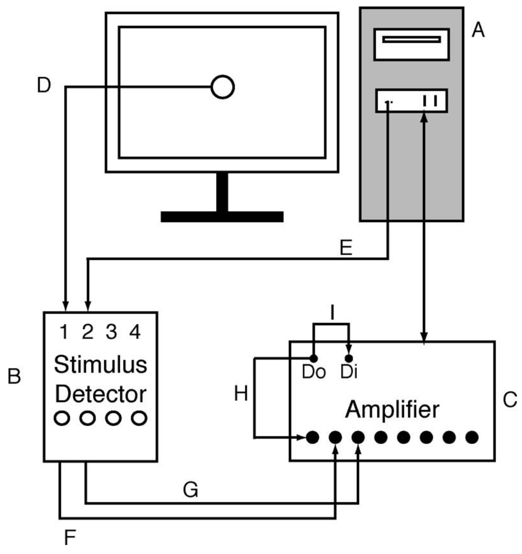

Fig. 2. Components of the validation system. (A) Computer running the BCI. (B) Stimulus detection trigger box. (C) ADC, with digital output (Do) and digital input (Di) ports. (D) Monitor and optical detector. (E) Audio output from computer. (F) and (G) outputs from trigger box to amp. (H) Amp digital output to analog channel input. (I) Amp digital output to digital input.

implementation was created using the BCI2000 system and the second implementation was created using a MATLAB-based BCI that implemented a simple mu-rhythm cursor movement task. Fig. 2 provides a reference hardware system; the components of this system will be described in subsequent sections.

1) Test Computers: We evaluated four different computer systems. These included two desktop computers and two laptops. The specifications for these computers are given in Table I.

Each computer ran Microsoft Windows XP with Service Pack 3; the MacBook Pro, and Mac Pro each ran Windows natively, and not in a virtual environment, such as VMWare or Parallels. Additionally, all tests were repeated on the Mac Pro with Microsoft Vista Enterprise Edition to compare performance across operating systems with identical components. In both operating systems, visual effects (e.g., Aero in Vista) were disabled to improve performance. Because drivers for the used ADC systems were only available for Windows, we did not test any other operating system.

2) Monitors: The method of generating the displayed image used by CRT screens is very different from that used in LCD displays. Therefore, to determine the effect that the monitor type has on display timing, all BCI2000-based tests were performed on both a CRT and LCD display. The CRT monitor resolution was set to 800 × 600 pixels, at a refresh rate of 100 Hz. The LCD resolution was set to 1024 × 768 pixels at a refresh rate of 60 Hz. The advertised typical response time for the LCD monitor was 16 ms, signifying the mean amount of time required for the digital image to be processed by monitor and for the liquid crystals to change states and let light pass through. CRT monitors have no such delay; instead, CRT monitors update the display by scanning a beam of electrons over a phosphorous-coated screen; when the beam hits the phosphorus, the screen fluoresces immediately. The beam scans the screen in rows, starting at the top-left of the screen, and ending at the bottom-right. Thus, the refresh rate of a CRT refers to the number of full scans completed in one second. In summary, the output latency for a CRT monitor will vary by a time that is inversely proportional to the screen's refresh rate.

3) Detection of Auditory/Visual Stimuli: We used a stimulus detection device, the g.TRIGbox from g.tec (Guger Technologies, Graz, Austria), to detect visual and auditory stimuli. To detect visual stimuli, an optical sensor was attached to the presentation monitor over the area in which the stimuli appeared. To detect auditory stimuli, we connected the audio output of the PC to the g.TRIGbox. Each input on the g.TRIGbox has a corresponding output, so that up to four stimuli can be detected simultaneously, each with an independent threshold setting. The threshold levels were manually adjusted for each stimulus, such that a stimulus (i.e., change in video luminescence or audio level) that exceeded the threshold was detected. When this occurred, the g.TRIGbox output a pulse on the corresponding output channel, which in turn was recorded by the ADC on an analog EEG channel.

4) ADC: We tested two different ADC systems. The first system consisted of two g.tec g.USBamp devices. Each device is capable of recording 16 channels and two digital channels at up to 38.4 kHz per channel. The analog inputs can record voltages in a range of ±250 mV. These voltages are amplified with a dc amplifier system and subsequently digitized with 24-bit resolution. The resulting digitized samples are transfered to a PC using a USB 2.0 connection. In our evaluations, we tested sampling rates up to 4.8 kHz (a sampling rate that is much higher than those typically used in EEG/ECoG recordings). No digital filtering was performed on the data prior to transmission to the PC.

The second system was the g.tec g.MOBIlab+ amplifier/digitizer. This device can record eight analog EEG channels and eight digital channels, sampled at 256 Hz per channel. The digitized samples are transferred to the PC via a RS232 serial interface or over a Bluetooth wireless connection. For this study, we used the RS232 serial interface. Initial tests with the Bluetooth connection revealed increased latency jitter over the serial interface, and an ADC latency of about 40 ms. According to the manufacturer, this latency is related to digital output buffering of the Bluetooth transmission, and not an input amplifier/digitizer delay. In the case of this particular amplifier and transmission protocol, an alternative means of generating an event pulse would be required.

5) Software: We evaluated the timing of two different BCI software packages. The first package consisted of the current version of BCI2000 v2.0. (no modifications were done to the software for our testing purposes. All programs were compiled with the Borland 2007 C++ compiler with all speed optimizations enabled.) The second package was an in-house MATLAB program that implemented a mu-rhythm cursor movement task. These MATLAB-based tests were executed on MATLAB 2009a for Windows.

Data collected from these two BCI packages were analyzed using a stand-alone analysis program that implemented the procedure described in this paper. This program is parameterized

TABLE I
LIST OF COMPUTERS TESTED

Computer	CPU Speed	RAM	Video Card
Mac Pro	Dual 2.8 GHz Quad-Core Intel Xeon	6GB	NVIDIA 8800GT (512MB)
Macbook Pro	2.2 GHz Intel Core2 Duo	4GB	NVIDIA 8600M (128MB)
Dell Optiplex 755	3 GHz Intel Core2 Duo	4GB	ATI Radeon 2600XT (256MB)
Dell Latitude 610	1.86 GHz Intel Pentium M	512MB	Integrated Intel 915M (16MB)

TABLE II
TASK PARAMETERS, THEIR TESTED VALUES, AND DESCRIPTIONS

	Sampling Rate (Hz)	# Channels Acquired & Processed	# Samples in one Block
Values	512, 1200, 2400, 4800	4, 8, 16, 24, 32	52, 120, 240, 480
BCI2000 Param. Names	SamplingRate	SourceCh	SampleBlockSize

using a a text file that describes how to analyze each test, e.g., which data channels contain ADC data, visual stimuli, and auditory stimuli. The software uses the BCI2000 binary data format, which contains the EEG data channels, task-related event information, and experimental configuration information, such as the sampling rate. In addition, we wrote a MATLAB script that converts MATLAB-based data into that data format. By modifying this script, data collected using *any* BCI system can be analyzed using our analysis program, provided that a minimum set of data specifications are met and saved. This validation and certification tool set is freely available as part of the BCI2000 distribution. Using this tool set, researchers can verify that any BCI system supports the timing requirements of their particular experiment, regardless of the specific system implementation, including hardware, operating system, or BCI software.

D. Task Configurations

As described, we tested two BCI software packages: BCI2000 and a MATLAB-based BCI. Using BCI2000, we tested several different common BCI tasks with different configurations. With the MATLAB-based BCI, a simple cursor movement task was written and tested in several different configurations.

It is important to emphasize that the MATLAB-based implementation was not primarily intended to serve as a comparison to the BCI2000-based implementation. Instead, we used the MATLAB-based BCI to demonstrate that the procedure developed in this paper can be applied to any BCI platform, irrespective of the operating system, and not just BCI2000; the data from the MATLAB BCI is saved in the MATLAB `*.dat` format, and is formatted and accessed completely differently than BCI2000 data. Furthermore, because the timing of some aspects of the MATLAB-based implementation were suboptimal, we demonstrate the potential utility of the techniques in this paper, as they highlighted these aspects of the MATLAB-based implementation that had unacceptably long latencies. It is quite possible that the timing of these aspects could be further improved (e.g., [16] and [17]), particularly when advanced MATLAB techniques using Java, Simulink, or the Real-Time Toolbox are employed.

1) BCI2000: The BCI2000 suite currently comes with three main feedback paradigms: a cursor movement task, a P300 speller task, and a generic stimulus presentation task. The cursor movement task can realize the movement of a cursor toward targets at programmable locations, similar to the classical “center-out” tasks used in the neuroscience literature (e.g., [7], [9], and [18]). The P300 speller task presents a matrix of characters or icons. Rows and columns of this matrix are flashed rapidly and randomly to elicit an evoked response (ERP) when the attended element is flashed, thus allowing the user to “select” elements from the matrix [19], [20]. Lastly, the stimulus presentation task presents a programmable series of auditory and/or visual stimuli to the user. This task can be used to elicit an ERP, such as the P300 response. We used the test procedure described in this paper to comprehensively test the protocols described earlier. We did this by systematically changing their most critical variables and determining the timing behavior of the resulting BCI system configuration; the test variables are described in Table II.

We evaluated latencies for each of the sampling rates and numbers of channels given in Table II, and for each of five BCI2000 configurations. This resulted in a total of 100 different tests. Specifically, for the cursor task, we compared two configurations (2-D and 3-D video output) that rendered the display differently. For the stimulus presentation task, we evaluated stimuli that gave video output, audio output, and combined video and audio output. Finally, for the spelling task, we used two configurations: one configuration displayed a 7×7 array of characters, and the other displayed a single character. The five configurations described earlier will, henceforth, be referred to as: cursor task (3-D), cursor task (2-D), P3 speller (7×7), P3 speller (1×1), and stimulus presentation (which included both auditory and visual stimuli). Thus, we evaluated a total of 100 BCI2000-based tests (i.e., five configurations, with four different sampling rates, and five different numbers of channels) for each computer. For any of these tests, if the average block duration was greater than the sample block size for two or more seconds (i.e., the computer could not keep up processing the sampled data in any two seconds period), the test stopped automatically and the next test began.

As shown in Fig. 3, the specific stimulus depended on the BCI2000 feedback paradigm (i.e., cursor task, P3 speller task, and stimulus presentation task). They were similar in that the stimulus always appeared in the same location. For the cursor task, a large white target appeared in the center of the screen, and then disappeared, leaving a black background. For the P3 speller task, a white letter or icon in the center of the screen was flashed

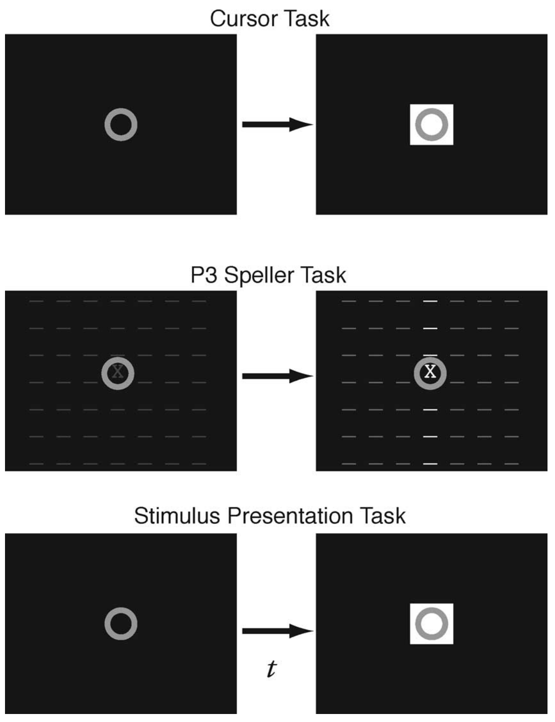

Fig. 3. Depiction of experimental flow. The red circle shows the location of the optical detector. For each test, a stimulus is presented at the same location repeatedly; the detector is input to the g.TRIGbox. When the light intensity at the location on the monitor (or audio volume for the audio-based tests) exceeded a threshold during the task, the trigger box output a 5 V pulse that was recorded by the data acquisition device.

on a black background. Similarly, for the stimulus presentation task, a white icon appeared in the center of the black screen. In the case that audio output was enabled, a short sound was played simultaneously with the icon. The peak amplitude of this sound stimulus occurred at 50 ms, but the input threshold was set to 0, so that any detectable sound stimulus would trigger an event. The noise floor of the stimulus detector was below threshold, and therefore, no false-positive events occurred. For all tasks, the stimulus was presented for 100 ms, with a pause interval of 100 ms, and a total of 50 times. The test suite was designed, so that once the visual and auditory detectors were setup correctly, no interaction with the experimenter was required.

Data from the acquisition system were processed as described next. First, the signals of each channel were referenced to a common-average reference (CAR). This is a commonly used spatial filter for EEG experiments in which the mean value of every channel is subtracted from each channel of interest. This procedure is implemented using matrix multiplication

$$C = B \times A \tag{6}$$

where A is the input signal with dimensions of $N \times S$ (N equals the number of channels and S equals the number of samples), B is the spatial filter and is represented as a $N \times N$ matrix, and the output signal C with dimensions equal to the input signal. The computational complexity of the implemented matrix–matrix multiplication is $O(N^3)$, indicating that the computational cost of this procedure is proportional to the number of elements (e.g., input channel or sample) to the third power.

These spatially filtered signals were then further processed depending on the specific task. The parameters for these procedures were chosen as representative of commonly used values during a typical experiment. For the cursor task, a power-spectral estimate using an autoregressive model of the input data was calculated for each channel. This was done using the autoregressive spectral estimation built into BCI2000 [21], [22], and a model order of 30 and window size of 500 ms. For the P300 spelling and stimulus presentation tasks, the system collected a 500 ms epoch of data after each stimulus presentation. Then, the system calculated the average of all collected epochs, separately for each stimulus and each channel. In a typical BCI experiment, the average is only calculated after a certain number of epochs are collected for each stimulus, e.g., 15 epochs. In our tests, the average was updated after every stimulus to determine an upper bound for the processing load.

2) MATLAB-Based BCI: The MATLAB-based BCI implemented a version of the BCI2000 cursor task (2-D) using MATLAB processing and visualization functions, and used the MATLAB equivalents of the algorithms employed in BCI2000. We used the $*$ operator to implement a spatial filter (i.e., for matrix–matrix multiplication), and the `pburg` function to calculate the power spectral estimate using the Burg algorithm. Additionally, we created a MATLAB software interface for the g.USBamp device, so that data could be acquired from the ADC directly into MATLAB. The cursor task was chosen for the MATLAB BCI because it uses the most computationally intensive signal processing algorithm. Finally, visual stimuli were presented using the MATLAB `figure` and `rectangle` functions, with OpenGL rendering enabled. The MATLAB figure and rectangle objects were created once, and the rectangle was displayed and hidden using the MATLAB commands `set(r, ''visible,'' ''on'')` and `set(r, ''visible,'' ''off'')`, respectively, where `r` was the handle of the rectangle object.

E. Analysis Methods

We implemented a validation and certification program to analyze and interpret the tests. The rising edges on three channels were detected to determine the time points for the events shown in Fig. 4. The associated software time-stamp values stored in the data file (i.e., the sample block onset and stimulus time) were used to determine the ADC latency, video and audio latencies, and system latencies. For example, if the optical detection pulse was on channel 2 (e.g., video (2) in Fig. 4), then the rising edges on that channel were compared to the starting time of the corresponding data block, and the difference between the two values was the video system latency for that block.

III. Results

This section presents example data for the tested BCI configurations using both BCI2000 and a MATLAB-based BCI, which demonstrate the type of results that can be obtained using the procedure shown in this paper; the results are not meant

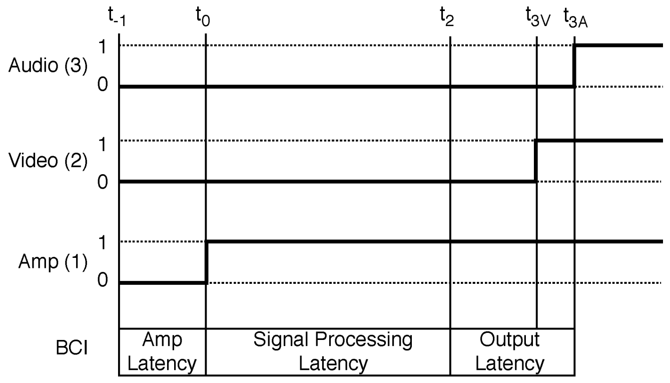

Fig. 4. Illustration of recorded events. Amp (1) records the digital output set after a block of data is acquired; at the same time, a software time stamp is stored (t_0). At t_2, the stimulus command is issued, and software time stamp is stored. Video (2) and audio (3) record the video and audio stimuli, respectively.

to be demonstrative of results that can be obtained only with BCI2000, or any other specific BCI implementation. Furthermore, all of the results shown are dependent on the specific hardware and BCI systems used, and are included to demonstrate that the procedure accomplishes what this paper claims. In fact, it is precisely *because* all BCI systems are different (and will, therefore, produce different results) that this procedure is even necessary.

The main results of the evaluations in this paper are shown in Fig. 5, which shows the latencies for the Mac Pro system running Windows XP with BCI2000. The results are shown for 60 tests, which include the three core BCI2000 tasks, repeated with all combinations of sampling rates and number of channels acquired. In order to condense the figure, the results for two of the tasks, cursor (2-D) and P3 speller (1×1), are not shown in Fig. 5, since they are similar to the cursor (3-D) and P3 speller (7×7) results, respectively (detailed descriptions for all tasks are provided later). The next paragraphs describe these results, compare these results obtained with the MATLAB-based BCI, and summarize the results for the other systems. For certain sets of configurations, we also tested the hypothesis that particular sets of latencies have the same mean and variance (resulting in a p-value for each comparison).

A. ADC Latency

1) g.USBamp: The latencies obtained from 60 tests for the g.USBamp ADC are shown graphically in Fig. 5 and in tabular form in Table III. An analysis of variance (ANOVA) determined that the type of the task had no effect on ADC latency ($p = 0.9$). However, the sampling rate and number of channels each had significant effects on ADC latency ($p < 0.01$). This is expected, since a higher sampling rate and/or channel count corresponds to more data that needs to be acquired by the ADC and transmitted to the PC.

The ADC latencies were not affected by whether BCI2000 or MATLAB were used. The MATLAB version of the data acquisition program was implemented in C++, and used the MATLAB MEX interface. Thus, it was not affected by any overhead that might be introduced by MATLAB.

Because the ADC latency was measured by analyzing the digitized signals, the resolution of the reported values was dependent on the sampling rate. Specifically, sampling rates of 512, 1200, 2400, and 4800 Hz correspond to a timing resolution of 1.95, 0.83, 0.42, and 0.21 ms, respectively. Therefore, a value of 0 ms corresponds to a latency that is smaller than a single sample at the given rate.

2) g.MOBIlab+: The g.MOBIlab+ ADC was tested using 16 channels (eight analog and eight digital) sampled at 256 Hz with the same five BCI configurations used for the g.USBamp. The mean latency for this ADC was 3.91 ms, equivalent to one sample at 256 Hz, and was the same for all five tasks. According to the manufacturer, this delay by one sample is due to internal buffers on the ADC.

B. Signal Processing Latency

1) BCI2000: As expected, the processing latency was significantly influenced by the sampling rate, number of channels, and task ($p < 0.001$). However, there were important similarities between configurations. The cursor (3-D) and cursor (2-D) processing latencies were nearly identical for all channel and sampling rate configurations, differing by no more than 0.5 ms, which is less than the resolution of the event timer (1 ms). Comparing the two P3 speller configurations, all processing latencies were significantly different ($p < 0.001$); the 7×7 configuration updated 49 averages with every stimulus presentation, compared to just one update for the 1×1 configuration, which had lower processing latencies in every configuration. Similarly, the stimulus presentation task only computed a single average for every stimulus, and had processing latencies that were similar to those for the P3 speller (1×1) configuration.

The two cursor task configurations used the most computationally intensive signal processing procedures. These procedures involved calculating the power spectrum of every channel for every sample block, and had a peak processing latency of 49.63 ms for both cursor tasks. The processing latencies for the cursor task (3-D) are shown in Table IV. The P3 speller task and the stimulus presentation tasks calculated the time average of 500 ms blocks of data for every stimulus, requiring considerably less computational power, and had a maximum processing latency of 24 ms.

We also calculated the correlation coefficients (r) between the total number of processed elements (i.e., the number of channels times the number of samples in a block) and the corresponding signal processing latency. The correlation for all tasks was $r > 0.97$ ($p < 0.001$), indicating that the processing latency was substantially influenced by the number of processed data elements, as expected.

2) MATLAB-Based BCI: The processing latency in the MATLAB-based BCI was also significantly influenced by the sampling rate and channel count ($p < 0.001$). In all cases, the processing time was longer in MATLAB than in BCI2000, except for four channels at 4800 Hz (7.29 ms in MATLAB versus 8.42 ms in BCI2000). At lower sampling rates (e.g., 512 and 1200 Hz), the processing latencies were much larger in MATLAB than BCI2000 (e.g., with 16 channels at 512 Hz, in

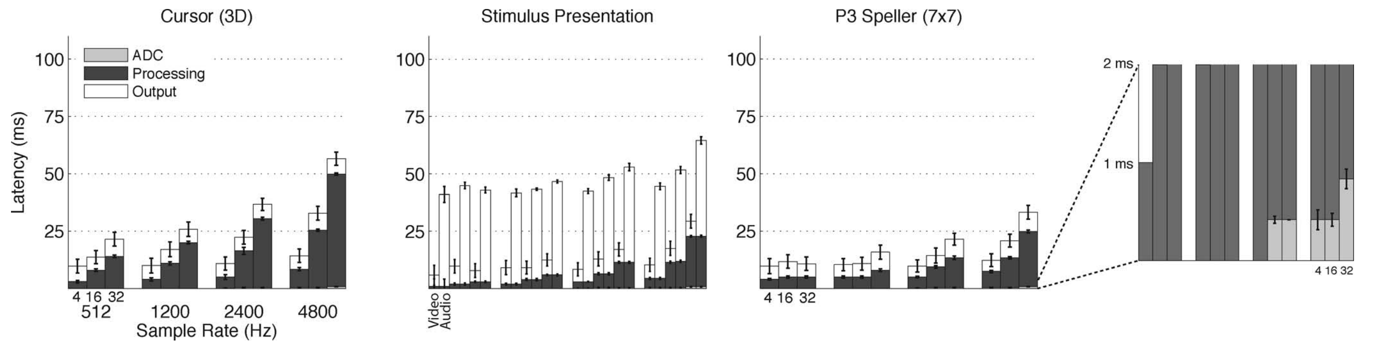

Fig. 5. System latency components for the three core BCI2000 tasks run on the Mac Pro computer running Windows XP. The stacked bars represent the contribution of each latency to the system latency, which is the height of each bar. Each group of three bars contains results for 4, 16, and 32 channels; each of these groups are organized by the sample rate (512, 1200, 2400, and 4800 Hz). The stimulus presentation task also contains audio latency values; in this case, the video latency and audio latency are adjacent. The right panel shows the values from 0 to 2 ms for the P3 speller (7 × 7) task to demonstrate the relative time scale of the ADC latency.

TABLE III
MEAN G.USBAMP ADC LATENCIES IN MILLISECOND FOR COMBINATIONS OF SAMPLING RATES AND CHANNELS

	Sampling Rate (Hz)			
# Chs	512	1200	2400	4800
4	0.00	0.00	0.00	0.42
8	0.00	0.00	0.42	0.42
16	0.00	0.00	0.42	0.42
24	0.00	0.00	0.42	0.62
32	0.00	0.00	0.42	0.83

TABLE IV
BCI2000 CURSOR TASK (3-D) PROCESSING LATENCIES IN MILLISECOND FOR COMBINATIONS OF SAMPLE RATES AND CHANNELS

	Sampling Rate (Hz)			
# Chs	512	1200	2400	4800
4	3.39 ± 1.59	4.25 ± 1.69	5.72 ± 1.86	8.42 ± 1.84
8	4.92 ± 1.60	6.46 ± 1.69	9.08 ± 1.92	14.28 ± 1.79
16	8.02 ± 1.52	11.08 ± 1.71	16.13 ± 1.78	25.60 ± 1.72
24	11.18 ± 1.52	15.29 ± 1.42	23.07 ± 1.93	37.60 ± 1.89
32	14.37 ± 1.56	20.43 ± 1.73	30.15 ± 1.79	49.63 ± 1.70

TABLE V
MEAN MATLAB CURSOR TASK PROCESSING LATENCIES IN MILLISECOND FOR COMBINATIONS OF SAMPLE RATES AND CHANNELS

	Sampling Rate (Hz)			
# Chs	512	1200	2400	4800
4	4.90 ± 0.30	6.47 ± 5.33	7.12 ± 5.53	7.29 ± 3.96
8	9.69 ± 0.46	11.61 ± 4.97	12.99 ± 5.74	15.03 ± 6.04
16	22.39 ± 7.64	23.90 ± 7.50	25.92 ± 8.06	28.76 ± 7.49

MATLAB the latency was 22.39 ms, while in BCI2000 it was 8.02 ms); however, as the amount of data increased, the processing latencies for MATLAB and BCI2000 were closer. This suggests an additional software overhead in MATLAB that is present regardless of the amount of data to be processed and unrelated to the actual algorithm used. As the amount of processed data increases, the algorithmic latency dominates the timing of this overhead, while with smaller amounts of data, the MATLAB overhead dominates the latency (see Table V).

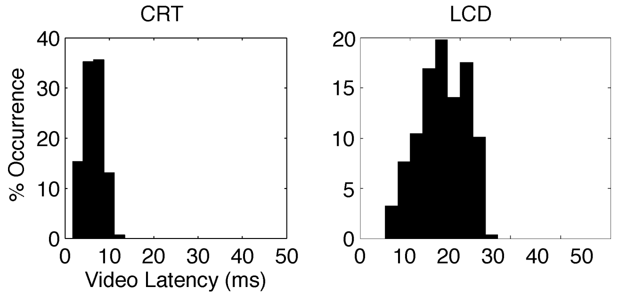

Fig. 6. Comparison of BCI2000 video output latencies for CRT and LCD monitors, for all tests.

C. Video Output Latency

1) BCI2000: In contrast to the signal processing latency, the video output latency did not depend on the number of channels, sampling rate, or task ($p = 0.67$). The mean video output latency on the MacPro Windows XP system using a CRT monitor with a refresh rate of 100 Hz was 5.06 ± 3.13 ms. The minimum and maximum output latencies were 1.33 and 11.33 ms, respectively. Because the current implementations of the BCI2000 feedback protocols are not synchronized to the refresh rate of the monitor, the video output latency values can range from 0 ms (i.e., when the output command is issued precisely at the monitor refresh) to the inverse of the refresh rate, Δt (i.e., when the output command is issued immediately following a refresh), which is 10 ms at a 100 Hz refresh rate. Our experimental results correspond closely to this: 1/(11.33–1.33 ms) = 100 Hz. The minimum output latency (1.33 ms) should then correspond to the latency of the system (operating system and video card) to process a graphics command and send it to the monitor.

As described, all tests were replicated using an LCD monitor (see Fig. 6). In this case, the mean video output latency was 15.22 ± 5.31 ms, with a range of 7.29 to 27.16 ms. The maximum possible refresh rate for this monitor was 60 Hz. The mean value is larger for the LCD monitor due to the "ON" time for liquid crystals, which is the amount of time required for the crystals to reconfigure and let light pass through when a current is applied [3]. This issue is addressed in more detail in the Section IV.

2) MATLAB-Based BCI: The MATLAB video output latency was very erratic compared to that obtained in BCI2000 using the same PC, algorithm, and (CRT) monitor. The mean latency was 45.80 ± 32.66 ms, with a range of 7.56 to 141.05 ms. This indicates that there is a large software overhead required to update the display, and that the exact time that the display is updated is very inconsistent from sample block to sample block. This suggests that the timing of stimulus presentation using MATLAB may be inadequate for many BCI applications.

D. Audio Output Latency

The stimulus presentation task in BCI2000, described in Section II-D, was used to measure the audio output latency. Initially, we ran the stimulus presentation task in three different configurations in which the video and audio latency were measured both separately, and then, together. However, we found that there was not a significant difference for the latencies if they are measured in the same task compared to if they are run separately. That is, the inclusion of auditory stimuli did not affect the video output latency, and *vice versa*.

The audio latency differed widely between computers and operating systems in the task that tested the audio output latency. The mean audio latency on the Mac Pro system running Windows XP was 40.45 ± 2.35 ms, and did not change based on the sampling rate or number of channels ($p = 0.67$). On the same computer running Windows Vista, however, the latency was 62.6 ± 4.59 ms, nearly 50% larger than the audio latency in Windows XP.

E. Block Duration and Jitter

1) BCI2000: The block durations were measured using both the software time stamps and hardware timing, as discussed in Section II. The block durations calculated using these two methods were compared to determine the accuracy of the software time stamp method versus hardware timing using pulses. We found that the two methods produced the same results for sampling rates of 1200, 2400, and 4800 Hz. When a sampling rate of 512 Hz was used, the software time stamps were unable to accurately measure the block duration, because the block duration was not an integer (i.e., the block duration was 101.56 ms at 512 Hz, compared to exactly 100 ms for the other sample rates). Since the software time stamp had a resolution of 1 ms, the block duration was measured as 102 and 101 ms on alternating blocks, and the mean block duration was measured as 101.56 ± 0.5 ms. Therefore, the hardware timing method was used for calculating the block duration.

The theoretical block duration for all tests was 100 ms, except for those with a sampling rate of 512 Hz, which had a block duration of 101.56 ms. The block duration was calculated from the rising edge of the digital input channel on consecutive sample blocks. This implies that the variance in the calculated block duration is dependent on the variance of the ADC latency. However, the block duration was not significantly dependent on the sample rate or number of channels ($p = 0.99$).

For tasks configured with sampling rates of 1200, 2400, and 4800 Hz, 94.6% of the measured block durations were exactly 100 ms; the standard deviation of the block durations (i.e., the jitter) was 0.46 ms, with a range of 99.58 to 100.42 ms. Hundred percent of the block durations for 512 Hz were 101.5625 ms (a block size of 52 samples at 512 Hz), resulting in 0 ms jitter. However, the jitter value was smaller than the temporal resolution given by the sampling rate.

2) MATLAB-Based BCI: The mean block duration was equal to the theoretical block duration for all sample rates and channel counts tested (i.e., 100 ms for sample rates of 1200, 2400, and 4800 Hz, and 101.5625 ms for 512 Hz). However, there was more jitter in the block duration in MATLAB than in BCI2000, i.e., the standard deviations of the block durations were larger. The jitter did not vary in a predictable manner with the number of channels or sampling rate, as shown in Table VI.

TABLE VI
MATLAB BLOCK DURATION JITTER

	Sampling Rate (Hz)			
# Chs	512	1200	2400	4800
4	0.49	3.48	5.88	0.21
8	0.49	2.41	1.95	6.49
16	6.41	5.91	11.80	5.28

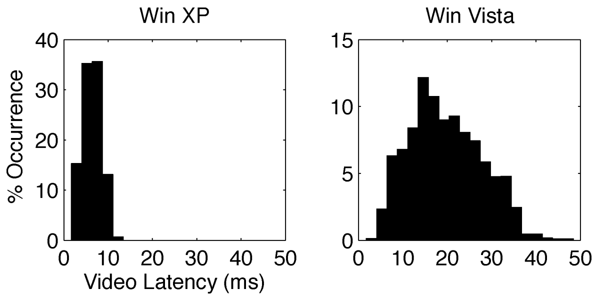

Fig. 7. Comparison of video output latencies for Windows XP and Windows Vista.

F. Operating System

In addition to the tests described earlier, all tests were replicated on the same Mac Pro system that dual-booted into Windows Vista Enterprise instead of Windows XP to determine the effect that the operating system has on the tasks using otherwise identical hardware. There were no significant measurable differences in the ADC latency or signal processing latency between Windows XP and Windows Vista for any task, sampling rate or number of channels ($p > 0.5$).

However, the video and audio output latencies for Windows Vista were significantly larger than those for Windows XP (each had $p < 0.001$). The mean video output latency was 20.26 ± 7.56 ms, with a range from 6.12 to 42.39 ms, compared to 5.72 ± 1.62 ms with a range of 1.33 to 11.33 ms in Windows XP. Fig. 7 shows the distributions of video output latencies for Windows XP and Windows Vista on the Mac Pro. The mean audio output latency in Windows Vista was 62.64 ± 7.55 ms, ranging from 52.58 to 112.91 ms, compared to a mean latency in Windows XP of 40.45 ± 2.32 ms. This suggests that the timing of stimulus presentation using Vista, at least with this particular hardware and driver configuration, may be inadequate for many BCI applications.

G. *Other Systems*

Results from other systems are briefly summarized here.

1) Dell Optiplex (Windows XP): The Dell Optiplex workstation had slightly better performance compared to the Mac Pro in every aspect. The most striking difference, however, was the audio output latency, which was significantly smaller (16.58 ± 1.62 ms) than for the Mac Pro (40.45 ± 2.35 ms). Additionally, the processing latencies were smaller on the Dell than on the Mac Pro for all tasks. This difference may be due to the Dell's higher clock speed (3.0 GHz for the Dell compared to 2.8 GHz for the Mac Pro; all processing algorithms were single-threaded, and thus, did not take advantage of the multiple cores in the Mac Pro).

2) Macbook Pro (Windows Vista): The processing latencies were longer for all tasks on this computer than for the Mac Pro running Windows XP or Vista, and than for the Dell Optiplex. This is likely due to the slower CPU clock speed (2.2 GHz) compared to the other two systems (2.8 and 3.0 GHz). The video output latency (19.72 ms) was nearly identical to the value obtained under Windows Vista on the Mac Pro (19.97 ms), indicating further that the operating system has a significant effect on the output latencies. Similarly, the audio output latency was 63.85 ms on this system, compared to 62.64 ms on the Mac Pro.

3) Dell Latitude (Windows XP): This system was the only one tested that was unable to maintain online performance in some tasks. Specifically, both of the cursor tasks performed poorly for higher sampling rates, particularly for the 3-D cursor setting, which had significantly longer processing latencies than for the 2-D cursor task. These longer processing latencies are presumably due to the integrated Intel graphics card, which cannot handle 3-D graphics; the processing latency includes some OpenGL preprocessing video commands, which are processed on the graphics card on the other systems, but are processed on the CPU on systems without hardware acceleration, such as this laptop.

The older and slower CPU clock speed also contributed to longer processing latencies for all tasks. However, it is also important to note that even this older laptop was able to process 32 channels sampled at 4800 Hz for the P300 and stimulus presentation tasks, and up to 32 channels sampled at 512 Hz in the 2-D cursor task. In most circumstances, a laptop such as this would likely be used as a portable EEG system, and would not require a sampling rate higher than 512 Hz. Additionally, usually only a subset of all channels are typically processed for a given task (e.g., C3, C4, and Cz for the cursor task), indicating that the computational requirements for many applications are not beyond those of many laptop systems.

IV. Discussion and Conclusion

This study presented a procedure to measure system latencies for any general hardware and software BCI configuration. We used this system to characterize the timing behavior of different BCI2000 and MATLAB-based BCI implementations. The results demonstrate that it is possible to accurately determine the latency at each step of the BCI system without the use of an external unit event timer or other hardware, besides a stimulus detector (a common, if not necessary, piece of equipment in many BCI and psychophysical laboratories). This procedure is general, so that it can be applied to different BCI systems, and that it can be modified or expanded for additional output events, e.g., the movement of a robotic arm. This procedure provides 1) the capability to determine if a computer is able of running a particular BCI configuration; 2) information about the performance capabilities for a given configuration (e.g., the minimum block duration at a particular sample rate and channel count); and 3) a method to test and optimize new modules (e.g., a new signal processing algorithm that is computationally intensive).

The first point is particularly important for experiments that include both audio and visual stimuli. For example, the Windows XP Mac Pro system had a mean video latency of 5.06 ms and a mean audio latency of 40.45 ms, a potentially large difference for psychophysical studies. Therefore, a better choice of computer in this case may be the Dell Optiplex computer, with video and audio latencies of 6.72 and 16.58 ms, respectively. Therefore, the choice of a computer system and components for use in a BCI experiment can have a significant impact on the results of the study, and there may be tradeoffs in the selected system.

Another important consideration in any psychophysical study including BCI research is the use of LCD monitors. LCD monitors operate with a fundamentally different technology compared to CRT monitors, and have become the standard display type shipped with new computers. They are more convenient in terms of size and weight, particularly, if the researcher is required to travel to the location of the subject instead of keeping the display in a fixed location in the laboratory. However, the increased overall latency and variability in output timing from LCD monitors may compromise the results. For example, a recent study showed that the onset of the P100 response in EEG is significantly delayed when different LCD monitors are used compared to a CRT monitor [1]. In summary, LCD monitors employ a more technically complex process to realize the video output. This latencies for this process are governed by a number of factors that include conversion of the digital signal to pixel-by-pixel values, setting the luminance of each pixel to a certain value (the response times for which can vary depending on the colors, e.g., the gray-to-gray response time can be very different from the black-to-white response time), the refresh rate, plus any additional image processing done on the monitor itself. An additional consideration is "overdrive" technology that is used in some LCDs, which can actually introduce an input lag of up to 60 ms, or inverse ghosting in which a shadow appears behind a quickly moving object, each of which could contribute to experimental problems.

The choice of operating system is clearly important as well. As shown in Fig. 7, video latency jitter, and therefore, system latency jitter was much higher for Windows Vista than for Windows XP on identical hardware and BCI software configurations. It is possible that the video card drivers were optimized for Windows XP, but not for Windows Vista, or that Windows Vista generally has inferior timing characteristics. Whatever the reason, the fact that the operating system itself can have a significant effect on BCI timing further demonstrates the need for the tools developed in this study.

The results also show that the choice of BCI system implementation can have a significant impact on the quality of the experiment and data recorded. In this study, the BCI2000 cursor movement task was replicated in the most recent version of MATLAB (2009a), including signal acquisition, signal processing, and application display. To most observers, both versions of the experiment would likely appear to run identically. However, the results obtained in this study show that the MATLAB BCI implementation had very inconsistent timing, particularly for updating the display. Such timing inconsistencies are likely to degrade the quality of the recordings and experiment.

As BCI experiments continue to push technical limits by moving toward experiments with higher sampling rates, channel counts, and experimental complexity, such as experiments using electrocorticographic (ECoG) electrodes and microelectrodes, it remains important to be confident that a particular experimental protocol can be executed properly on a chosen system."As evidenced by the results shown in this paper, even a relatively high-end computer, the 8-core Mac Pro, may be unable to maintain online performance with just 32 channels at a sampling rate of 4.8 kHz. (It is important to remember here that the signal processing demands of this configuration were relatively high. Simple data acquisition and stimulus presentation, or processing with less demanding configurations, proved possible at these data rates on this computer.)

The methods developed in this study provide a means of identifying and addressing system bottlenecks, allowing the researcher to decide whether the capabilities of a system are sufficient for a given experiment, if the system needs to be upgraded, or if algorithms need to be further optimized.

References

[1] A. M. Husain, S. Hayes, M. Young, and D. Shah, "Visual evoked potentials with CRT and LCD monitors: when newer is not better," *Neurology*, vol. 72, no. 2, pp. 162–164, Jan. 2009.

[2] J. H. Krantz, "Tell me, what did you see? The stimulus on computers," *Behav. Res. Methods, Instrum., Comput.*, vol. 32, no. 2, pp. 221–229, May 2000.

[3] N. Stewart, "Millisecond accuracy video display using OpenGL under Linux," *Behav. Res. Methods, Instrum., Comput.*, vol. 38, no. 1, pp. 142–145, Feb. 2006.

[4] C. D. Chambers and M. Brown, "Timing accuracy under Microsoft Windows revealed through external chronometry," *Behav. Res. Methods, Instrum., Comput.*, vol. 35, no. 1, pp. 96–108, Feb. 2003.

[5] W. J. MacInnes and T. L. Taylor, "Millisecond timing on PCs and Macs," *Behav. Res. Methods, Instrum., Comput.*, vol. 33, no. 2, pp. 174–178, May 2001.

[6] B. Myors, "Timing accuracy of PC programs running under DOS and Windows," *Behav. Res. Methods, Instrum., Comput.*, vol. 31, no. 2, pp. 322–328, May 1999.

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

[8] J. R. Wolpaw and D. J. McFarland, "Control of a two-dimensional movement signal by a noninvasive brain-computer interface in humans," *Proc. Nat. Acad Sci. USA*, vol. 101, no. 51, pp. 17849–17854, Dec. 2004.

[9] G. Schalk, K. J. Miller, N. R. Anderson, J. A. Wilson, M. D. Smyth, J. G. Ojemann, D. W. Moran, J. R. Wolpaw, and E. C. Leuthardt, "Two-dimensional movement control using electrocorticographic signals in humans," *J. Neural Eng.*, vol. 5, no. 1, pp. 75–84, 2008.

[10] J. A. Wilson, E. A. Felton, P. C. Garell, G. Schalk, and J. C. Williams, "ECoG factors underlying multimodal control of a brain-computer interface," *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 14, no. 2, pp. 246–250, Jun. 2006.

[11] D. M. Taylor, S. I. H. Tillery, and A. B. Schwartz, "Direct cortical control of 3D neuroprosthetic devices," *Science*, vol. 296, no. 5574, pp. 1829–1832, Jun. 2002.

[12] 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.

[13] J. Mellinger, G. Schalk, C. Braun, H. Preissl, W. Rosenstiel, N. Birbaumer, and A. Kubler, "An MEG-Based brain-computer interface (BCI)," *Neuroimage*, vol. 36, no. 3, pp. 581–593, Jul. 2007.

[14] J. D. Bayliss, "Use of the evoked potential P3 component for control in a virtual apartment," *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 11, no. 2, pp. 113–116, Jun. 2003.

[15] G. Edlinger and C. Guger, "Laboratory PC and Mobile Pocket PC brain-computer interface architectures," in *Proc. Conf. IEEE Eng. Med. Biol. Soc.*, 2005, vol. 5, pp. 5347–5350.

[16] G. Müller-Putz, R. Scherer, C. Brauneis, and G. Pfurtscheller, "SSVEP-based BCI," *J. Neural Eng.*, vol. 2, pp. 123–130, 2005.

[17] C. Guger, S. Daban, E. Sellers, C. Holzner, G. Krausz, R. Carabalona, F. Gramatica, and G. Edlinger, "How many people are able to control a P300-based brain–computer interface (BCI)?" *Neurosci. Lett.*, vol. 462, no. 1, pp. 94–98, 2009.

[18] A. B. Schwartz, R. E. Kettner, and A. P. Georgopoulos, "Primate motor cortex and free arm movements to visual targets in three-dimensional space. i. relations between single cell discharge and direction of movement," *J. Neurosci.*, vol. 8, no. 8, pp. 2913–2927, Aug. 1988.

[19] E. Donchin, K. M. Spencer, and R. Wijesinghe, "The mental prosthesis: Assessing the speed of a P300-based brain-computer interface," *IEEE Trans. Rehabil. Eng.*, vol. 8, no. 2, pp. 174–179, Jun. 2000.

[20] E. W. Sellers and E. Donchin, "A P300-based brain-computer interface: Initial tests by ALS patients," *Clin. Neurophysiol.*, vol. 117, no. 3, pp. 538–548, Mar. 2006.

[21] D. J. McFarland and J. R. Wolpaw, "Sensorimotor rhythm-based brain-computer interface (BCI): Model order selection for autoregressive spectral analysis," *J. Neural Eng.*, vol. 5, no. 2, pp. 155–162, Jun. 2008.

[22] D. Krusienski, D. McFarland, and J. Wolpaw, "An evaluation of autoregressive spectral estimation model order for brain-computer interface applications," in *Proc. Conf. IEEE Eng. Med. Biol. Soc.*, Aug. 30–Sep. 3, 2006, vol. 1, pp. 1323–1326.

J. Adam Wilson (S'02–M'10) received the B.S. degree in electrical and computer engineering from Ohio State University, Columbus, OH, in 2003, and the M.S. and Ph.D. degrees in biomedical engineering from the University of Wisconsin-Madison, Madison, WI, in 2005 and 2009, respectively.

He is currently a Postdoctoral Fellow in Department of Neurosurgery, University of Cincinnati, Cincinnati, OH. His research interests include studying mechanisms of cortical spreading depression and neuroprostheses.

Jürgen Mellinger received the M.S. degree in physics from Tuebingen University, Tuebingen, Germany, in 1999.

Since 2001, he has been a Research Associate at the Institute of Medical Psychology and Behavioral Neurobiology, University of Tuebingen. Since 2004, he has also been a Lead Developer in the brain–computer interface 2000 software project.

Gerwin Schalk (M'04) received the M.S. degree in electrical engineering and computer science from Graz University of Technology, Graz, Austria, in 1999, the M.S. degree in information technology, and the Ph.D. degree in computer and systems engineering from Rensselaer Polytechnic Institute, Troy, NY, in 2001 and 2006, respectively.

He is currently with New York State Department of Health, Wadsworth Center, Albany, NY. He has authored or coauthored 39 peer-reviewed journal publications and has given around 100 invited lectures world-wide. His research interests include engineering, scientific, and commercial aspects of devices that interface the brain with external devices for the purpose of communication or diagnosis. His work has been extensively featured by the media, including features on CNN, NBC, and CBS, articles in *Technology Review, Wired, New Scientist,* etc. He is also listed in *Who's Who in the World* and *Who's Who in America*.

Dr. Schalk was the recipient of several awards.

Justin Williams (M'01) received the B.Sc. degree in mechanical engineering and engineering physics, in 1995 and 1996, respectively, from South Dakota State University, Brookings, and the M.Sc. and Ph.D. degrees in bioengineering from Arizona State University, Tempe, in 2001.

From 2002 to 2003, he was jointly appointed a Research Fellow in the Department of Biomedical Engineering, University of Michigan, Ann Arbor, and a Research Scientist in the Department of Neurological Surgery, University of Wisconsin-Madison, where he was involved in neurosurgery applications of neural engineering devices. He is currently an Assistant Professor in the Department of Biomedical Engineering, University of Wisconsin-Madison. His research interests include biomicroelectromechanical systems, neural interface development, and neuroprostheses.