High-performance execution of psychophysical tasks with complex visual stimuli in MATLAB

Wael F. Asaad, Navaneethan Santhanam, Steven McClellan and David J. Freedman

J Neurophysiol 109:249-260, 2013. First published 3 October 2012;
doi: 10.1152/jn.00527.2012

Journal of Neurophysiology publishes original articles on the function of the nervous system. It is published 24 times a year (twice monthly) by the American Physiological Society, 9650 Rockville Pike, Bethesda MD 20814-3991. ESSN: 1522-1598. Visit our website at http://www.the-aps.org/.

J Neurophysiol 109: 249–260, 2013.
First published October 3, 2012; doi:10.1152/jn.00527.2012.

High-performance execution of psychophysical tasks with complex visual stimuli in MATLAB

Wael F. Asaad,[2] Navaneethan Santhanam,[1] Steven McClellan,[1] and David J. Freedman[1]
[1]*Department of Neurobiology, The University of Chicago, Chicago, Illinois; and* [2]*Brown Institute for Brain Science, Norman Prince Neurosciences Institute and the Department of Neurosurgery, Alpert Medical School, Brown University and Rhode Island Hospital, Providence, Rhode Island*

Submitted 19 June 2012; accepted in final form 2 October 2012

Asaad WF, Santhanam N, McClellan S, Freedman DJ. High-performance execution of psychophysical tasks with complex visual stimuli in MATLAB. *J Neurophysiol* 109: 249–260, 2013. First published October 3, 2012; doi:10.1152/jn.00527.2012.—Behavioral, psychological, and physiological experiments often require the ability to present sensory stimuli, monitor and record subjects' responses, interface with a wide range of devices, and precisely control the timing of events within a behavioral task. Here, we describe our recent progress developing an accessible and full-featured software system for controlling such studies using the MATLAB environment. Compared with earlier reports on this software, key new features have been implemented to allow the presentation of more complex visual stimuli, increase temporal precision, and enhance user interaction. These features greatly improve the performance of the system and broaden its applicability to a wider range of possible experiments. This report describes these new features and improvements, current limitations, and quantifies the performance of the system in a real-world experimental setting.

behavior; methodology; neurophysiology; psychophysics; vision

THE ABILITY TO CONTROL PRECISELY the presentation of sensory stimuli to subjects and to monitor their behavioral responses is critical for performing experiments in a wide range of research fields from social and cognitive psychology to systems and cognitive neuroscience. Furthermore, such experiments may utilize a diversity of model organisms, ranging from insects and rodents to nonhuman primates and human subjects. Although the exact needs of different experimental approaches can differ widely, all studies investigating perception, cognition, or action of an awake subject share a number of common requirements. First, researchers need control over the sensory stimuli delivered to the subject. Second, researchers need the ability to monitor and respond to subjects' actions. Third, such research requires the ability to analyze behavioral data both in real-time and in offline analysis. Finally, when working with subjects engaged in complex behaviors, the ability to create quickly and easily a wide range of tasks and to modify task parameters rapidly is essential.

A number of software tools for controlling such experiments have been developed over the years, virtually all of which were designed to run on a specific and narrow range of hardware and software, and most required the user to program in a relatively low-level programming language (e.g., C or C++). One limitation of this approach is that not all experimentalists arrive at the laboratory with a strong programming background. If they do have a programming background, they may not be familiar with the particular language or platform required by the experiment control software used in that laboratory. In a number of research fields, including neurophysiology, cognitive neuroscience, and perceptual psychophysics, MATLAB has become a nearly ubiquitous tool that many investigators use extensively for data analysis. MATLAB is a high-level programming language that emphasizes ease of use and portability at the expense of performance (i.e., processing speed) and access to low-level hardware functions (e.g., memory management). In the past, the idea of running experiments, which require millisecond-level timing precision for presenting stimuli and recording responses, in a high-level programming environment such as MATLAB was not feasible because the performance of the software was inadequate for applications requiring millisecond-level temporal performance.

Recent improvements both in computing hardware and MATLAB software have made it possible to achieve temporally precise control over experiments using MATLAB (Asaad and Eskandar 2008a; Brainard 1997; Gardner and Larsson 2012; Kleiner et al. 2007; Rose et al. 2008; Zeki 2012). Previous reports described efforts to create a flexible and full-featured tool for controlling behavioral and neurophysiological experiments (Asaad and Eskandar 2008b) in MATLAB. At that time, there were a number of limitations to both the features and performance of the software that made it inadequate for some applications. Here, we present a number of performance enhancements and describe new features of the software that greatly improve its effectiveness for controlling a wide range of behavioral, perceptual, and neurophysiological studies in both human and nonhuman subjects.

MATERIALS AND METHODS

Our tested system was composed of a Dell OptiPlex 980 computer with an Intel i5-680 processor running at 3.60 GHz containing 4 gigabytes of random access memory (RAM; Dell, Round Rock, TX). The operating system was 64-bit Microsoft Windows 7 Ultimate (Microsoft, Redmond, WA) running with a non-Aero desktop theme. The graphics hardware in this machine consisted of two ATI Radeon HD 3400 cards with 1,979 megabytes of available video RAM. The two outputs from one of the graphics cards were mirrored and connected to two displays, each running in full-screen mode at a resolution of 1,280 × 1,024 pixels and a refresh rate of 75 Hz, and video was double buffered. One of the displays was a 21-in. Hewlett Packard P1110 CRT monitor, which was the subject's display. The other display was a 19-in. Dell 1908FP LCD running at 1,280- ×

Address for reprint requests and other correspondence: D. J. Freedman, Dept. of Neurobiology, The Univ. of Chicago, 947 E. 58th St., MC0928, Chicago, IL 60637 (e-mail: dfreedman@uchicago.edu).

1,024-pixel resolution and 75 Hz and provided the experimenter with a duplicate of the subject's display. The second video card was connected to a 22-in. Dell P2210 LCD monitor running at 1,680- × 1,050-pixel resolution and 60 Hz and served as the experimenter's display. The experimenter's display window was set to update about every 50 ms during behavioral monitoring to allow continuous observation of the subject's performance.

MATLAB software (32-bit version R2010b; The MathWorks, Natick, MA), including the Data Acquisition Toolbox (version 2.17) and the Image Processing Toolbox (version 7.1), was used to write the behavioral control code tested here and to analyze the data reported here. All functions comprising our software were written as simple .m files that are user-editable. MATLAB was run in the default nonmultithreaded mode and with the Java Virtual Machine disabled ("matlab -nojvm" from the command prompt). MATLAB figures for the experimenter's display (created using built-in graphics and plotting functions in MATLAB) relied on OpenGL with hardware acceleration enabled.

For controlling visual stimuli presented on the subject's display, two distinct software solutions were used and evaluated separately in the tests described here. In the first solution, both static images and dynamic movie stimuli were displayed using the XGL toolbox (version 1.2), a set of low-level routines for video control (based on DirectX from Microsoft), which were generously provided by Jeffrey S. Perry at the University of Texas at Austin (Perry 2012). In the second solution, images and movies were displayed using calls to Psychophysics Toolbox (PTB; Version 3), an OpenGL-based MATLAB toolbox (Brainard 1997; Kleiner et al. 2007; Pelli 1997; http://www.psychtoolbox.org). The software routes all calls to the video display through a single gateway function. Separate versions of this function were created to use either the XGL toolbox or PTB routines for graphics. Apart from the differences in these calls to the graphics system and different approaches for loading and preprocessing visual stimuli, all aspects of the two versions of the software were identical.

An optimized system profile was created as described previously (Asaad and Eskandar 2008a) to minimize the amount of processor time that could be taken by other applications. The task was run at the highest priority level (priority can be set in the software main menu) allowed by Windows ("Real Time"), and the priority was automatically lowered to "Normal" during the intertrial intervals (ITIs) to allow other pending activities time to execute.

Behavioral signals were monitored using two identical National Instruments PCI-6221 data acquisition (DAQ) boards (National Instruments, Austin, TX). Each of the multifunction DAQ cards was connected to a National Instruments BNC-2090A breakout box using 6.0-ft shielded connector cables. These were interfaced in MATLAB through the Data Acquisition Toolbox. Although the Data Acquisition Toolbox is not intended for real-time control, prior tests have demonstrated that these routines provide sufficient performance for submillisecond precision for behavioral monitoring (Asaad and Eskandar 2008a). We also performed tests to compare the latency of DAQ operations between National Instruments' PCI and USB DAQ boards. We used the USB-6009 device for this purpose.

The incoming behavioral signals were split into two analog DAQ input boards to allow for more rapid sampling and simultaneous storage of the data. As described previously (Asaad and Eskandar 2008a), logging and sampling data from the same board is subject to upload delays caused by temporary storage of samples in the local memory buffer of the board. Our software is able to detect the presence of two identical boards and will allocate one for storage and another for online, unbuffered sampling.

Digital event markers were sent to an external neural DAQ system (Plexon MAP system; Plexon, Dallas, TX) using digital input/output channels from one of the two National Instruments PCI-6221 cards. Event markers were encoded with nine digital lines corresponding to an 8-bit word and one strobe/trigger digital line. It was reported previously that temporal performance using digital lines on the DAQ card showed poor temporal performance compared with using the parallel port (Asaad and Eskandar 2008a). However, we have found substantially improved temporal performance of digital outputs (faster than 1.0 ms per strobed word) of several PCI-based National Instruments cards when using newer versions of MATLAB and newer computing hardware (see *Performance*).

Our tests were run using nonhuman primates (rhesus macaque monkeys, *Macaca mulatta*) as subjects. For the data and analyses presented here, monkeys performed either visual discrimination or visual categorization tasks that required animals to maintain gaze fixation and to indicate their decisions using a manual touch bar. For our animal subjects, fluid reward (fruit juice) delivery was controlled via analog or digital output of a transistor-transistor logic (TTL) pulse trigger to an external device that operated a solenoid valve that controlled the flow of juice delivered to the animal. All procedures and protocols were approved by the University of Chicago's Animal Care and Use Committee and were in accordance with National Institutes of Health guidelines.

An optical eye-tracking system (EyeLink 1000; SR Research, Ontario, Canada) produced analog horizontal and vertical signals conveying information about eye-gaze position with a temporal resolution of 1.0 kHz. A touch-bar detection circuit was used to determine whether the subject made contact with a manual touch bar, and the output of this circuit was a voltage that could take one of two states (high or low) depending on whether the subject made contact with the lever. The eye signals were acquired with a sampling rate of 1.0 kHz via two analog input channels (corresponding to horizontal and vertical eye position) on each of the two DAQ cards.

To assess the performance of our software for movie presentation, we examined data acquired during the ongoing training of a subject. The monkey was trained to perform a delayed match-to-sample task with 360° of motion directions. Twelve motion directions were used as sample and test stimuli. Each trial began with the onset of a 0.5° square that the monkey had to fixate for 500 ms. Following the fixation period, a sample stimulus (1 of the 12 motion directions) was shown for 650 ms followed by a memory delay (1,000 ms) and a test stimulus (1 of the 12 motion directions) for 650 ms. The monkey had to release a manual touch bar if the test was the same direction as the sample. If the test was a nonmatch (on 50% of trials), it was followed by an additional delay (200 ms) and a 2nd test stimulus (650 ms) that was always an exact match to the sample and required a lever release. Stimuli were high-contrast, random-dot movies composed of 256 dots per frame that moved at 5° per frame. The movie files ($n = 145$ AVI movie files) were generated before the behavioral session, contained 90 frames each, were 155 × 155 pixels in size, and encoded with 24-bit color depth. There were a total of 972 conditions in the condition file for the task. Monkeys maintained gaze fixation within a 2.0° radius of a fixation point during the fixation, sample, delay, and test periods of the task.

To assess the performance of MonkeyLogic for image presentation (and to provide a comparison with the previous report), we also examined data acquired during the training of a subject to perform a categorization task using static images. The monkey was familiarized with a group of 25 images over a period of several months and then trained to distinguish them from novel stimuli during a delayed match-to-category (DMC) task with novel and familiar images as categories. In the DMC task, each trial began with the monkey fixating on a central fixation spot for 500 ms followed by the presentation of a sample stimulus (either a novel or a familiar image) for 650 ms. The sample was followed by a delay of 1,000 ms and then a test stimulus (either a familiar or novel image) for 650 ms. If the test stimulus belonged to the same category (i.e., novel or familiar) as the sample, the monkey had to release a lever to obtain a juice reward. If the test stimulus was not the same category as the sample, the monkey was required to hold through another delay period (75 ms) followed by a second test stimulus (presented for 650 ms) that was of the same category as the sample and required a lever release. The monkey was

also trained to perform an image-dimming detection task. In this task, the monkey had to release a touch bar in response to a subtle reduction in the luminance of an image stimulus. Trials started with 500-ms fixation followed by the presentation of an image for 500 ms plus a random time (exponential with a mean of 500 ms). This was followed by a subtle dimming of the image. The monkey had to detect and report immediately the dimming of the image by releasing the touch bar to obtain a juice reward. The stimuli for the task were 100- × 100-pixel 24-bit color RGB images downloaded from the Internet and resized in MATLAB. The fixation spot was 0.5°, and the monkey was required to fixate within ±2.0°.

The software performance is defined by the extent to which it can display stimuli, collect behavioral responses from subjects, and output digital signals to neural DAQ or other devices with a high degree of temporal reliability and accuracy. The software is split into several components that are responsible for these functions. A previous report (Asaad and Eskandar 2008b) measured the temporal performance of several functions in the software that are used to display stimuli on the screen ("toggle"), collect behavioral responses ("track"), and output event markers to a neural DAQ ("eventmarker"). Functions are divided into "entry," "core," and "exit" sections. Entry time refers to the amount of time required for initialization of each function before the execution of the essential activity of that function. Core time is the amount of time taken by the function to execute the activity. Exit time is the time taken to perform cleanup activity and return control to the user once the core activity has been completed. Although both entry time and exit time are important for evaluating the performance of the function, the core section must execute rapidly to ensure that we meet our goal of submillisecond precision for detecting and responding to behavioral events. Two other functions are also especially important from a performance perspective: the "ITI" function is used to prepare stimuli for the next trial and the "trial" function that initializes other trial-related functions at the start of each trial and stores trial data at the end of a trial. We also measured and report the ability of the software to run trials involving movies, an important new feature that has been added since the previous report.

For the toggle subfunction, entry time refers to the time required to parse the user's command options, determine the type of stimulus to display (image, movie frame, etc.), and clear previous visual stimuli if necessary. Core time refers to the act of displaying the stimulus on the screen. The exit time is the time required to display the control screen updates corresponding to the currently visible stimulus before returning control to the user. For the track subfunction, entry time refers to the time required to parse the user's command options and calculate the target thresholds required for detecting changes in behavior (as specified by the user's call to this function). The core activity consists of retrieving the latest analog data samples, transforming them into calibrated coordinates, and comparing these coordinates against those of possible targets. This section of the function loops as fast as the system will allow to monitor the subject's behavior until a user-specified amount of time has elapsed. Since a majority of the time in most trials is spent during this loop, the performance of this loop largely determines the responsiveness of the software to behavioral events. Exit time corresponds to the time required to remove targets from the control screen and return control to the user's script. The entry time of the eventmarker subfunction is the time required to parse the user's command options and convert digital codes into binary for outputting on the digital lines. The core time is the time required to write the digital bytes to the neural DAQ system (2 operations are required: 1st setting the value of the marker being sent and then the strobe bit). During exit, the strobe bit is reset, the time stamp is buffered for local storage, and control is then returned to the user's script.

For the trial function, the entry time is the time required to initialize all video and input-output subroutines (including toggle, track, and eventmarker, among others), begin DAQ, and start the trial timer. The core time of this depends entirely on the user's script and is thus not shown here. Exit time refers to the time taken from the end of the user's script to the end of the trial, during which analog data and event markers are written to the local data file (event marker codes were also sent to a separate neural DAQ system in real-time). During the preparation time for the ITI, the next trial is selected according to the prespecified block- and condition-selection functions (as specified by the user). Stimuli are then loaded from disk to video buffers, and the control screen is updated to provide the latest behavioral performance measures (overall percentage correct, block percentage, condition percentage, etc.) and information about the software performance (average cycle rate and slowest cycle during trial). The loading of stimuli constitutes the bulk of this time.

RESULTS

In following sections, we will first describe feature additions, updates, and enhancements to the software followed by a detailed quantitative evaluation of the temporal performance of the software by analyzing real behavioral data collected during behavioral training sessions. Next, we provide details about the compatibility of the software with various hardware and software configurations. Finally, we discuss existing limitations to the software and future plans for feature additions that will further enhance performance and usability.

Design and Features

PTB. A key change to the software is that the DirectX-based XGL toolbox video routines (see MATERIALS AND METHODS), which were used previously to control the presentation of all graphics on the subject's display, can now be replaced entirely with PTB (see MATERIALS AND METHODS) depending on user preference. This has been a frequently requested change from users of the software because of its wide adoption among vision researchers, active development and support community, and potential to use the many features of PTB for generating and displaying both simple and complex stimuli. Furthermore, PTB runs on Windows, Macintosh, and Linux operating systems, opening up the possibility that our software could be ported to other operating system platforms in the future.

At the current stage of development, the adoption of PTB for controlling the video system (and several keyboard-related features) has been achieved with relatively minimal impact to the functionality or performance of the software, and the usability of the software has not changed from the user's perspective. In the future, new features will be added to our software to take advantage of additional features of PTB. Detailed information about the relative performance of the XGL and PTB versions of our software is presented in the *Performance* section of this paper.

Displaying movies. The ability to display movies is a major new feature of the software and allows complex and dynamic stimuli to be presented to the subject with careful control over the precise timing of stimulus onset, offset, and synchronization of movie frames with the refresh of the video system. The ability to display movies is available in both the XGL toolbox and PTB versions of the software.

In both the XGL and PTB versions of the software, the approach taken to displaying dynamic stimuli in our software is to present frames of previously generated AVI movie files. The primary advantage to this approach is that the computation

required to generate the movie itself is accomplished outside of our software and before running an experiment. This allows stimuli to be rapidly loaded, processed, and displayed without causing unacceptable delays during a trial or experimental session. One limitation is that it does not currently easily allow new movies to be generated while a behavioral task is running (e.g., during the ITI or during the trial itself), and the contents of a movie file cannot be easily manipulated while a movie is being played. However, several simple manipulations of movies can be achieved while running a behavioral task. For example, frames of a movie file can be played in any order (e.g., forward, backward, or in an arbitrary or random frame sequence), and the movie (or any static image) can be translated along any arbitrary path (even while a movie is playing). For tasks that require subjects to pursue a visual target smoothly, the software now allows setting a translating stimulus as a target for fixation.

The overall approach to displaying movies in the XGL and PTB versions of the software is nearly identical. Movie support was initially added to the XGL version (before development of the PTB version of the software), and PTB development was achieved primarily by identifying functions in PTB for movie-related operation that were equivalent to the XGL functions used for movie functionality in the original XGL version. These include functions for copying video data to the video buffer, the back buffer of the screen, the flip operation, and clearing the screen and video buffers in preparation for the next frame. During the ITI, before each trial, the movie frames for all movies in the upcoming trial are loaded into video RAM on the video card. To display these movies during the trial, the software controls the onset of the first movie frame (as specified by the user's trial function), the display of subsequent movie frames, and the offset of the last movie frame with high temporal precision. Because both the XGL toolbox and PTB relay precise timing information about the status of the video system (e.g., which raster line is currently being drawn), the drawing of each movie frame is synchronized to the video refresh interval. This allows movie frames to be drawn precisely at a predetermined rate without artifacts such as tearing and without skipping frames. In the very rare event of a skipped frame because of an unexpected delay or error in the user's timing code, a visible warning is displayed on the experimenter's display and is also saved as a time-stamped behavioral code in the behavioral data file. Missed frames were not encountered during the tests described in here. The rate of movie frame display can be specified by the user. For example, movie frames can be advanced and displayed on each video frame or on every n video frames, where n is an integer. In all of the tests presented here, the video refresh rate was set to 75 Hz, and movie frames were advanced on each video refresh.

It is essential that the processing overhead associated with the display of movies does not interfere with the precise ability to monitor and respond to behavioral events such as breaks of gaze fixation, release of a lever, change in status of a button, or movement of a joystick. This is achieved by incorporating the function calls to display the next video frame into the same track routine that monitors analog inputs such as eye position, joystick, and buttons. If a movie had been drawn to the display before the call to track, the appropriate time to display the next frame of the movie is determined by monitoring the trial timer, raster line, and frame number that is currently being drawn by the video system. At the appropriate time (before the refresh), the toggle routine is called from within track to display the next movie frame. Because the main track loop runs much faster (typically >1,000 Hz) than the video refresh rate (typically 60–100 Hz), behavioral monitoring performance in track is minimally affected by the movie-related code on most cycles of the loop. There is a modest decrease in track performance on the cycle in which toggle is called (internally, by the track function) to draw the next movie frame (see *Performance*). However, this performance decrease is relatively small and likely acceptable for most applications.

There are several notable limitations to movie functionality at present. First, as described above, new movie files cannot be generated during the ITI or during trial execution. Instead, movie data must be loaded from pregenerated AVI movie files. Second, movie frames are currently not loaded directly from the AVI movie file at the start of each trial. Instead, each AVI movie file is "preprocessed" before the start of an experimental session. The preprocessing step loads the video data from the AVI file, transposes the data into the format expected by the video system, and saves the data as a MATLAB format data file. This dramatically improves the speed of loading movies at the start of each trial but adds an additional step (and sometimes a delay) before starting a trial that uses many movies. The loading time for trial-relevant stimuli is subtracted from the ITI, and delays only occur when this time exceeds the duration of the ITI. When many movie frames or very-high-resolution movies need to be displayed on a trial, the time required for loading the movie file and copying these data to video RAM increases and can cause extended ITI durations. See the *Performance* section for information regarding strategies for improving movie-loading performance.

Usability features. A number of new functions and features have been added for the purpose of improving the stability, usability, and versatility of the software. Many of these features are aimed at giving the user more rapid control of the parameters of the behavioral task to facilitate training of the subject. For example, customizable "hot keys" have been added for changing task parameters (e.g., reward amount and task-interval durations) or executing simple functions (e.g., giving a reward or centering the eye-gaze position) without pausing a trial during task performance. A customizable popup window has also been added to allow the task to be paused and task parameters to be modified without having to quit and restart the behavioral task. In addition, the ability to display custom plots on the control screen has been added to give the user control over the information that is displayed on the experimenter's display during a session.

Several features have been added to facilitate the process of creating and troubleshooting new behavioral tasks. A "simulation mode" has been implemented that allows the user to bypass the analog inputs and manipulate the eye position, joystick position, and button status variables using the keyboard. This has proven to be useful for testing and debugging newly created behavioral tasks and components of the experimental rig. A second useful debugging tool is a "user text" box that has been added to the experimenter's control screen. The user text box can output customized text generated from the timing file and is useful for viewing the status of variables in the timing code, condition numbers, and task parameters.

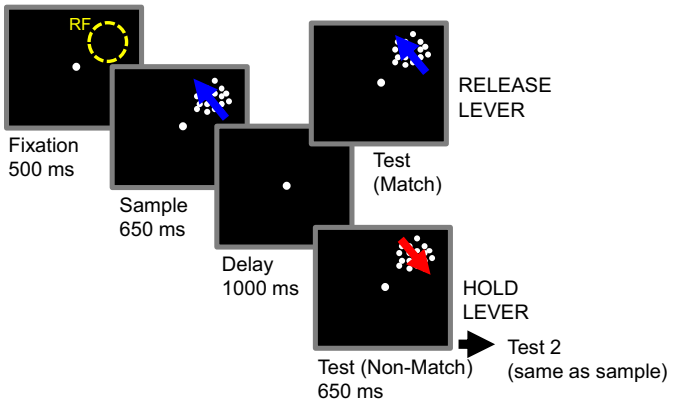

Fig. 1. The behavioral task used to study movie performance. The subject performed a delayed match-to-sample task and had to indicate (by releasing a lever) whether sample and test stimuli were identical matches. If the sample and test were not identical, the subject had to wait until a 3rd stimulus was presented, which was always an identical match to the sample (and required a lever release). Sample and test stimuli were both high-contrast, random-dot movies on a gray background. The location of the receptive field (RF) of a neuron is indicated by the dotted yellow circle.

Finally, the stability of the software has been substantially improved. Mouse and keyboard input (except for the keyboard hot keys) are now disabled while a behavioral task is running. This greatly reduces the likelihood that the focus of the operating system will be removed from MATLAB (which would otherwise have caused an unrecoverable crash of the behavioral task). In the event of a crash, changes have been made to the routines for saving the behavioral data file so that the file can be loaded and the behavioral data recovered even in the event of a serious crash of the behavioral task, MATLAB, or the operating system.

Compatibility. A goal for the development of this software is to maintain its compatibility with the widest possible range of computer and experimental hardware and software. Currently, the software is compatible with personal computers (PCs) running the Windows operating system (including 32- and 64-bit versions of Windows XP, Vista, and 7). At present, there are no immediate plans to port the software to Macintosh or Linux as the MATLAB Data Acquisition Toolbox (which handles all analog and digital input and output in our system) is only available for MATLAB running Windows. The software is compatible with older (R2007b and higher) and more recent (MATLAB R2010a) versions of MATLAB (32-bit only, which can be run on top of a 64-bit Windows operating system). The deprecation of non-Java plotting routines in more recent versions of MATLAB (R2011a and up), however, cause cosmetic problems in the display of the user menu; strategies to fix this limitation are currently under development.

Performance

Performance during movie presentation. A key new feature is that the software is now capable of displaying pregenerated movies as stimuli. We demonstrate here that movie performance meets the criterion of submillisecond precision (for a description of the task used, see MATERIALS AND METHODS and Fig. 1). We present a comparison between the current temporal performance of the software while displaying movies and a previous version displaying static image stimuli using the functions listed above. Advances in PC hardware, operating system software, MATLAB, and DAQ hardware and drivers have allowed us to improve the performance of most of these functions. In Table 1, we show that the performance of several core routines in the software that are used to initialize trial functions (trial), display movie stimuli (toggle), collect behavioral data from the subject (track), send digital codes to a neural DAQ system (eventmarker), and prepare stimuli for display during the trial (ITI). Functions are divided into entry, core, and exit sections (for more details, see MATERIALS AND METHODS). These values are also compared between versions of the software that use XGL or PTB to control video functions.

One area where there has been a dramatic improvement in performance is during the exit time of toggle, which has been reduced from 25 to 0.54 ms in the XGL version and 0.58 ms in the PTB version. This is a notable improvement that will be of practical use for users as this virtually eliminates a period during which the system was unresponsive following a toggle call, as described in a previous report (Asaad and Eskandar 2008b). The core time of toggle shows a marked difference between the XGL and PTB versions. The XGL version runs in 0.10 ms on average (an improvement over the 0.18 ms in the previous version of the software), but the PTB version requires 1.9 ms to run. This is a potential source for concern as the toggle function is responsible for displaying stimuli and must run efficiently to ensure that there are no timing delays. In the case of movies, this is a more serious problem as there are far more toggle calls during trials using movies than static stimuli. Nevertheless, we have not encountered practical issues in the presentation of movie stimuli using either XGL or PTB versions of the software, although a goal for future development is to improve this performance and ensure that the presentation (flip) operation always occurs in <1 ms. Eventmarker, which is now optimized for sending digital signals via the digital output lines of the DAQ board (rather than through the parallel port as discussed in the previous report), has reduced the time required to perform entry and exit operations by an order of

Table 1. *Software performance during tasks with movies*

	Static Images: Asaad and Eskandar (2008b), ms	Movies: XGL, ms	Movies: PTB, ms
Trial			
Entry	0.11	13.94	16.29
Exit	14.07	12.73	12.85
Toggle			
Entry	1.02	1.03	0.18
Core	0.18	0.10	1.90
Exit	25.85	0.54	0.58
Track			
Entry	1.15	1.93	1.93
Core	0.98	0.56	0.67
Exit	1.09	0.21	0.21
Eventmarker			
Entry	0.24	0.04	0.02
Core	0.5	0.59	0.60
Exit	0.01	0.0042	0.0038
Intertrial interval			
Preparation time	99.51	415.1702	406.46

Values are means. The measured times for Asaad and Eskandar (2008b) represent data collected over 1,601 trials. XGL toolbox and Psychophysics Toolbox (PTB) values for the current study are obtained over 3,400 and 2,600 trials, respectively. For details, refer to MATERIALS AND METHODS.

magnitude (to 0.04 and 0.02 ms in the XGL and PTB versions, respectively) while maintaining submillisecond precision for its core operation (Table 1).

One routine that shows substantially reduced timing performance (but increased functionality) is the trial entry section, which runs once before the start of each trial. Trial entry is responsible for initializing all DAQ, video, and eventmarker subroutines. The increase in processing time for this function is due to its increased functionality, as it now supports more types of stimuli (e.g., movies), and new subroutines have been added to allow greater control over stimuli and task parameters. The primary subroutine responsible for the increase in trial entry time is track initialization (Table 1), which initializes the control screen and verifies that data can be acquired by the user-specified behavioral inputs. Consistent with the increase in the number of operations performed by this function, the entry time for track has nearly doubled, from 1.05 to 1.93 ms (in both XGL and PTB versions). Trial exit, on the other hand, has shown a decrease in the time required to store all acquired behavioral data, event markers, and associated time stamps to the data file.

The track function, during which analog signals are obtained and transformed into calibrated coordinates, has seen an increase in entry time (as discussed above) but has shown improvements in the core and exit times. The core of this function is responsible for determining whether a task-relevant behavioral event (such as an eye movement, joystick movement, or a button press) has occurred and also controls the updating of movie frames (through calls to toggle) during movie display. As this operation is a main factor in determining the responsiveness of the software to behavioral events, it is critical that the core of the track function is run as quickly as possible to ensure that updates to the stimuli occur in a timely manner.

For tasks involving the display of movies, the ITI shows increased processing time, consistent with the additional processing requirements for movies. During the ITI, preprocessed movie files are loaded into memory from disk and then buffered onto the video device. The increased processing times are primarily due to loading the preprocessed movie files (which contain the movie data) from disk. Because ITIs are often 1 s or longer during behavioral tasks, the increased processing time for movies is typically not a limitation in practice (except perhaps in cases where very short ITIs are desired or when many movies, movies with many frames, or very-high-resolution movies are used). For users that need to load very large amounts of movie data in the ITI (which might cause the ITI to exceed the desired duration), several options are available for improving the speed of movie loading. Preliminary testing with a solid state drive (SSD) for storage of the preprocessed movie files improved performance substantially (by a factor of 3 on 1 tested PC), and preliminary testing suggests that substantial further performance gains beyond the SSD may be obtained by using a RAM disk for storage of movie data. For the performance data presented here, movie data were stored on a conventional hard disk. Users also have the option of loading all movie data (for all trials) at the beginning of a session (assuming sufficient memory capacity of the video card) rather than loading movie data for each trial in the ITIs. This can result in a delay before the start of the first trial, but subsequent ITIs are fast since no additional movie data need to be loaded. Additional performance gains can be obtained in some situations by disabling file compression when saving preprocessed movies. Finally, future support for movies with a lower bit depth (e.g., 8- or 16-bit) may be added that would further improve the movie-loading performance.

Reliability of video timing. The reliability of video timing during movie playback in both the XGL and PTB versions of the software was verified by measuring and analyzing the output of a photodiode attached to the subject's display (Fig. 2). To do so, a movie consisting of alternating "white" and "black" frames was displayed for ~650 ms (50 total frames) using a black background color on the display. During these tests, a photodiode circuit was used to indicate (with an increase in voltage) precisely the time at which white frames appeared on the screen. The movie was located in the center of the display, and the photodiode was fixed to the screen near the center of the display. The voltage output of the photodiode was recorded as an analog input in the software, and the timing of the photodiode signal was compared with the timing of the stimulus-on eventmarker code in offline analysis. In both the XGL (Fig. 2*A*) and PTB (Fig. 2*B*) versions of the software, the first movie frame (which was white) appeared ~8.0 ms after the stimulus-on eventmarker, which is consistent with the 13.33-ms raster period of the display (at 75 Hz) and the location of the photodiode near the midpoint of the display. No repeated or

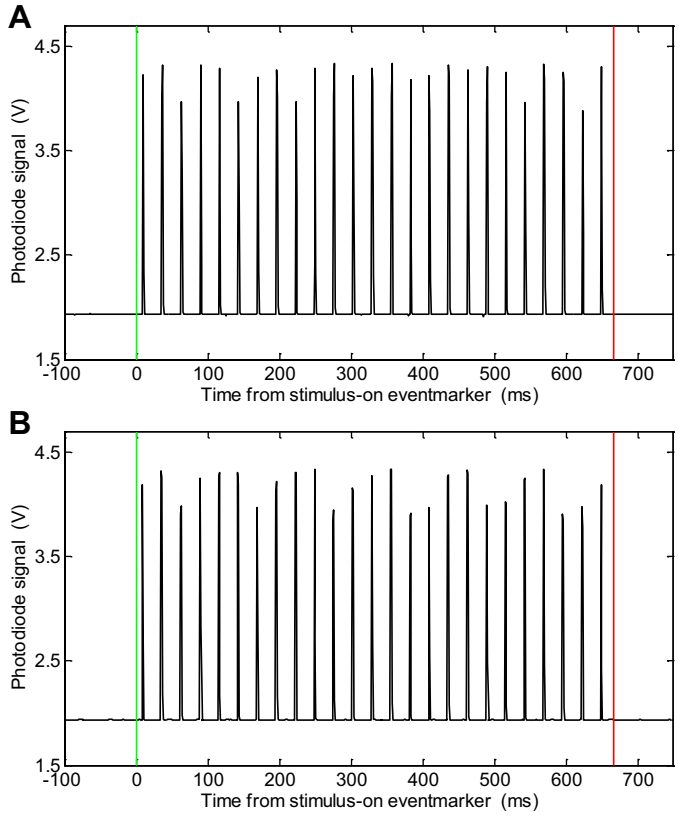

Fig. 2. Photodiode verification of video performance during movie presentation in the XGL (*A*) and Psychophysics Toolbox (PTB; *B*) versions of the software. The black trace indicates the voltage recorded from the output of a photodiode circuit with the photodiode sensor affixed to the display screen (near the center of the display). The green and red lines indicate the timing of the stimulus-on and stimulus-off event markers, respectively. The movie consisted of alternating "white" and "black" frames, and frames were played at a rate of 75 Hz. The interval between the green line and 1st upward deflection of the photodiode signal was ~8.0 ms, which is consistent with the frame rate and the position of the photodiode near the center of the display.

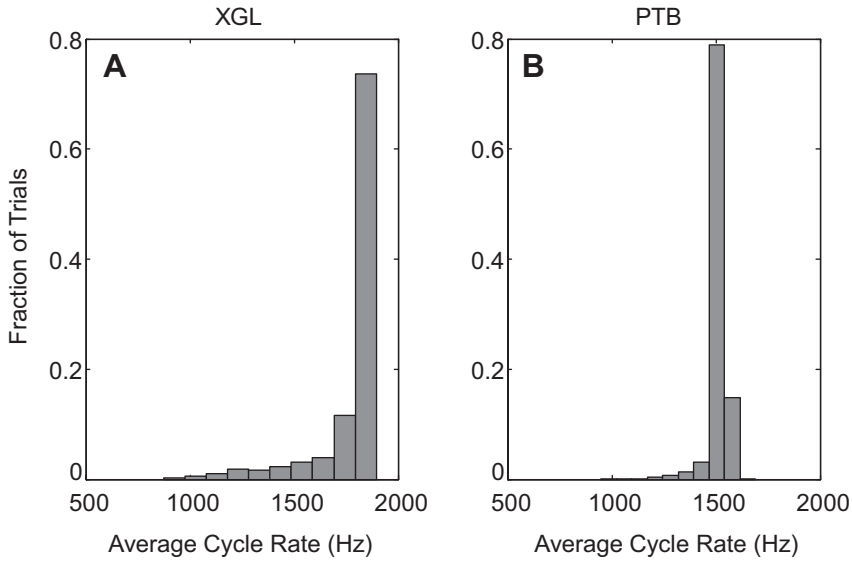

Fig. 3. Histogram of average cycle rates across all movie trials. *A*: the average cycle rate across 3,400 movie trials using XGL was 1,767 Hz. The average cycle rate during the 1st trial was 1,785 Hz, nearly double the 913 Hz reported for images in the previous report (Asaad and Eskandar 2008b) despite having additional processing requirements. 99% Of trials had an average cycle rate >1,100 Hz. 0.32% Of trials had average cycle rates <1,000 Hz, and the lowest average cycle rate across all trials was 873.5 Hz. *B*: the average cycle rate across 2,600 movie trials using PTB was 1,503 Hz. The average cycle rate during the 1st trial was 1,431 Hz. 99% Of trials had an average cycle rate >1,200 Hz. 0.04% Of trials had average cycle rates <1,000 Hz, and the lowest average cycle rate across all trials was 948.5 Hz.

dropped frames were encountered during the example trials shown in Fig. 2 or in the remainder of the testing session with the photodiode. Note that the photodiode responds on only every other frame, when pixels go from black to white when the white frame is drawn. The regular pattern of photodiode responses and absence of responses during the black frames indicates that the video system is appropriately advancing through the movie frames without repeating or skipping frames.

Behavioral monitoring. A key measure of the performance of the software is its rapid ability to monitor and respond to changes in the status of analog inputs. To measure this performance, we examined the cycle rate of the track subfunction, which runs as a loop to monitor values in the analog input buffers for script-specified events (e.g., release of a lever or change in eye position) that may require a response from the software (e.g., turning on or off a stimulus, delivering a reward, etc.). We present histograms of average cycle rates during movie trials using XGL and PTB (Fig. 3). Cycle rates primarily reflect the temporal performance of the track routine core loop. Higher cycle rates indicate that the software can respond more rapidly to behavioral events such as a change in target acquisition or a response of the subject. Cycle rates have increased substantially since the previous version of the software (Asaad and Eskandar 2008b), with mean cycle rates across all trials of 1,767 Hz using XGL and 1,503 Hz using PTB (compared with 960 Hz using XGL in the 2008 study). The performance of the software allows millisecond-level precision in behavioral monitoring except on cycles where movie stimuli are manipulated or during calls to toggle when using the PTB stimulus presentation framework, which may take 1–2 ms (see Table 1). We also computed the average cycle latency for all cycles of track during the task involving movies (Fig. 4). Cycle latency is defined as the time required to complete one cycle of the track function. Across all cycles, average cycle latency was 0.56 ms in the XGL version and 0.71 ms in the PTB version. Average cycle latency during movie presentation for those cycles in which movie frames were updated (through an internal call to toggle) was 1.88 ms in the XGL version and 3.30 ms in the PTB version.

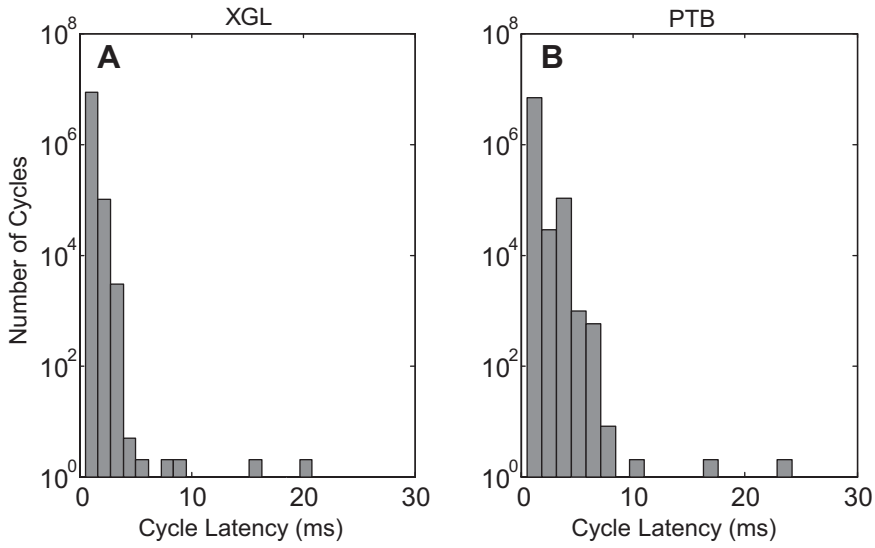

Fig. 4. Histogram of cycle latencies across all trials. *A*: average cycle latency across all 3,400 trials for XGL was 0.56 ms. 99.9% Of cycles had a latency of <2 ms. Here, we exclude 2 cycles with latency 101 ms (0.00002% of all cycles). *B*: average cycle latency across all 2,600 trials for PTB was 0.71 ms. 99.9% Of cycles had a latency of <5 ms.

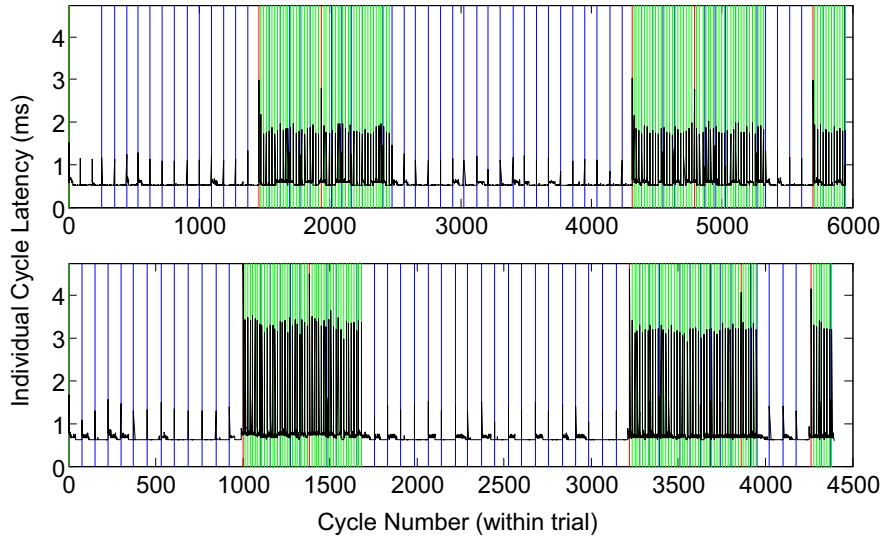

Fig. 5. Cycle latencies for a typical behavioral tracking epoch on a single example trial using XGL (*top*) and PTB (*bottom*) stimulus presentation frameworks. During these example trials, 2 inputs (eye position and button status) are being monitored by the track routine, and movie stimuli are displayed. The black trace indicates individual cycle latencies for all cycles in the trial. The green vertical lines indicate cycles during which the toggle subfunction is called. The blue vertical lines indicate cycles in which the control screen is updated. The red vertical lines indicate cycles in which the control screen is updated and the toggle subfunction is called.

During a trial, to monitor the subject's behavior, the track subfunction performs processes such as monitoring analog input buffers (for changes in target acquisition, fixation, response, etc.), updating the experimenter's control screen, and toggling stimuli on the subject's screen. In Fig. 5 (*top*, XGL; *bottom*, PTB), individual cycle latencies are plotted during a typical behavioral tracking epoch during which analog input values were monitored, the experimenter's control screen was updated, and movie frames were updated. The green, blue, and red vertical traces in the figure represent cycles during which various operations took place within the track subfunction: monitoring 2 analog inputs (1 with X- and Y-components, comprising a total of 3 analog lines) and toggling stimuli onto the screen (green), monitoring 2 analog inputs and updating the experimenter's control screen (blue), and doing all 3 together (red). In this particular trial, 3 movie stimuli were presented, as indicated by the regions with green vertical bars and increased cycle latencies. We grouped cycles across all trials during which these processes occurred and observed their impact on individual cycle latency (Fig. 6). The bars show the latency of

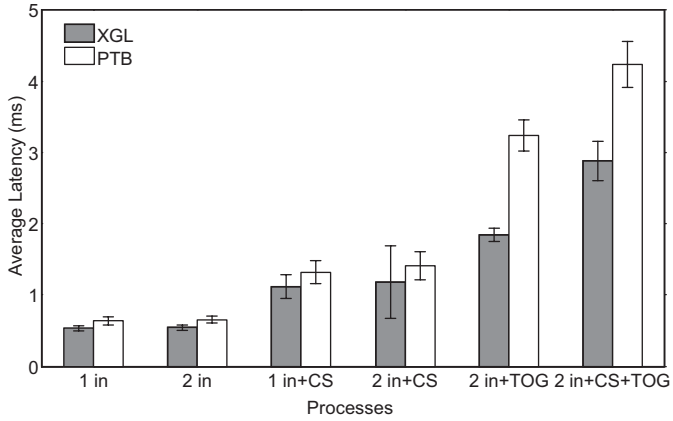

Fig. 6. Average latency of processes in the track function. The bars show the time required for different combinations of behavioral inputs, and video operations in track using XGL (gray) and PTB (white) stimulus presentation frameworks are shown. Each process consisted of monitoring 1 or 2 inputs ("in") while track simultaneously performed updates to the experimenter's control screen ("CS") and toggled movie stimuli on the subject's screen ("TOG"). Error bars represent standard deviation.

different combinations of monitoring one or two behavioral inputs ("in") and performing video operations such as updating the experimenter's control screen ("CS") or toggling movie stimuli ("TOG") on the subject's screen. Tracking two analog inputs rather than one analog input did not have a substantial impact on cycle latency; however, updates to the control screen and toggling stimuli onscreen corresponded with increased cycle times. Overall, XGL showed somewhat better temporal performance than PTB, especially when stimuli are toggled on the subject screen. This is also evident from the observation that the core time of the toggle function in XGL runs more quickly than its counterpart in the PTB version, as discussed above and shown in Table 1.

Performance during static-image presentation. To compare directly the current versions (PTB and XGL) of the software and the version (XGL only) from the previous report (Asaad and Eskandar 2008b), we ran a similar task to that used in the previous report (for description, see MATERIALS AND METHODS and Fig. 7) and measured the software performance. Improvements were seen in many trial-related subfunctions (Table 2), including the exit time of toggle, the core time of track (which runs repeatedly over the course of the trial), and across all areas of eventmarker (except in the core section for the PTB version). The core time of toggle has seen a modest decrease in performance from 0.18 to 0.5 ms in XGL but is still in line with the goal of submillisecond precision. However, in the PTB version, the core section of toggle takes longer to execute (2.15 ms). The entry time of the trial function has seen a substantial decrease in performance, compared with the older version, as explained above. The ITI function has shown a moderate decrease in performance from 99.51 ms in the previous version to 148.89 ms in the new XGL version and 152.17 ms in the PTB version. We measured the average cycle rate across all trials during this task in both the XGL and PTB versions (Fig. 8). The average cycle rate in the XGL version was 1,883 Hz, nearly double the average cycle rate from the previous report, which was 960 Hz. The average cycle rate for trials in the PTB version was 1,435 Hz, also a marked improvement over the previous version.

Digital output performance. At the time of the previous report (Asaad and Eskandar 2008a), it was recommended that

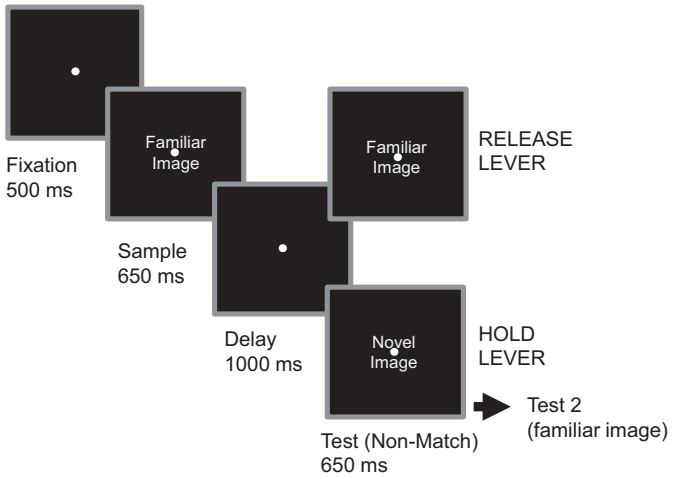

Fig. 7. The behavioral task used to study image performance. The subject performed a delayed match-to-category task and had to indicate (by releasing a lever) whether sample and test stimuli were of the same category, novel or familiar. If the sample and test were not identical, the subject had to wait until a 3rd stimulus was presented, which was always the same category as the sample (and required a lever release). Sample and test stimuli were both full-color RGB images.

the parallel port was used to transmit behavioral codes (as 8-bit digital words plus a strobe bit) or other digital signals to external devices (e.g., a neural DAQ system). Since that time, updates to the MATLAB Data Acquisition Toolbox have dramatically improved the performance of digital outputs on the DAQ board, which are now suitable for the purpose of transmitting time-sensitive digital outputs. Thus it is now recommended that digital inputs and outputs make use of the DAQ device. Here, we present a comparison of the timing of analog and digital input and output functions for a PCI DAQ card and a USB DAQ device (Table 3).

PCI cards outperformed the USB device on all four functions ("putsample," "getsample," "putvalue," and "getvalue"). These functions are critical to the software performance. Getsample is used during trials to obtain analog inputs in the track function and typically is called several thousand times per trial. Putsample is used by the reward function to send an output signal to an external reward device if connected via an analog input. The putvalue function is used to generate TTL signals and output digital event codes to the neural DAQ system and also in the reward function when using a digital output. The USB device showed slower performance, requiring >1.0 ms to return, rendering them (currently) unsuitable for our purposes. For example, the getsample call, without which analog inputs cannot be acquired, takes 14.3 ms to run over USB, which is much too slow for our requirements. By contrast, the same getsample call was completed in 0.66 ms using the PCI DAQ card.

DAQ performance, issues, and mitigation. One current limitation that we have encountered during testing is that, in certain situations, the number of analog samples acquired during the course of a trial is sometimes slightly fewer than the number of samples expected based on the length of the trial. For example, when sampling at a rate of 1 kHz, the number of samples should equal the length of a trial in milliseconds. However, a problem arises because of the way the samples are made available to the software through the MATLAB Data Acquisition Toolbox. Samples, although acquired at the correct sample rate by the DAQ card, are not immediately available to the MATLAB workspace. Rather, they are made available to MATLAB in discrete chunks. MATLAB Data Acquisition Toolbox relates the number of samples available through a property known as SamplesAvailable. From one millisecond to the next, if SamplesAvailable has not changed, this means that, even though a sample may have been acquired and buffered by the DAQ card itself, it will not be available within MATLAB until the next chunk is transferred from the DAQ buffer. Once the trial has ended, any samples currently on the buffer of the DAQ card are discarded and not made available to MATLAB. Thus the number of samples recorded for that trial will be less than expected due to the missing samples (typically <200) at the end of the trial. Importantly, this issue does not affect the real-time monitoring of analog behavioral inputs (e.g., eye or joystick signals) because that functionality is handled by the 2nd, unaffected DAQ board running in an ad hoc (nonlogging) mode (Asaad and Eskandar 2008a).

To demonstrate this discrete chunks phenomenon, we compared time stamps from the "toc" MATLAB operation (to establish time since the start of DAQ) with the size of SamplesAvailable over the course of 2,500 ms using a sample rate of 1 kHz. If the size of the buffer is updated on every sample acquisition, we should see both increasing linearly. However, we found that SamplesAvailable increased in a stepwise fashion rather than linearly in the case of toc. This indicates that SamplesAvailable lags the time stamp and is updated only at finite intervals when new chunks of data are made available. Figure 9 illustrates this phenomenon. For the DAQ device that we tested, we found that the chunk size was a multiple of 64 (74% of chunks were of size 64, 25% were of size 128, and 0.52% were of size 192). Based on reports from other laboratories, this value can vary depending on the hardware configuration being used.

To ensure that the software provides all samples acquired during the course of a trial, we modified the "end_trial"

Table 2. *Software performance during tasks with static images*

	Asaad and Eskandar (2008b), ms	Images: XGL, ms	Images: PTB, ms
Trial			
Entry time	0.11	13.08	16.47
Exit time	14.07	10.87	13.84
Toggle			
Entry time	1.02	1.21	0.26
Core time	0.18	0.55	2.01
Exit time	25.85	0.47	0.50
Track			
Entry time	1.15	2.14	2.15
Core time	0.98	0.53	0.71
Exit time	1.09	0.27	0.31
Eventmarker			
Entry time	0.24	0.03	0.01
Core time	0.5	0.0029	0.58
Exit time	0.01	0.0021	0.0038
Intertrial interval			
Preparation time	99.51	148.89	152.17

Values are means. The measured times for Asaad and Eskandar (2008b) represent data collected over 1,601 trials. The measured times for the current version using XGL and PTB represent data collected over 2,400 and 1,500 trials, respectively.

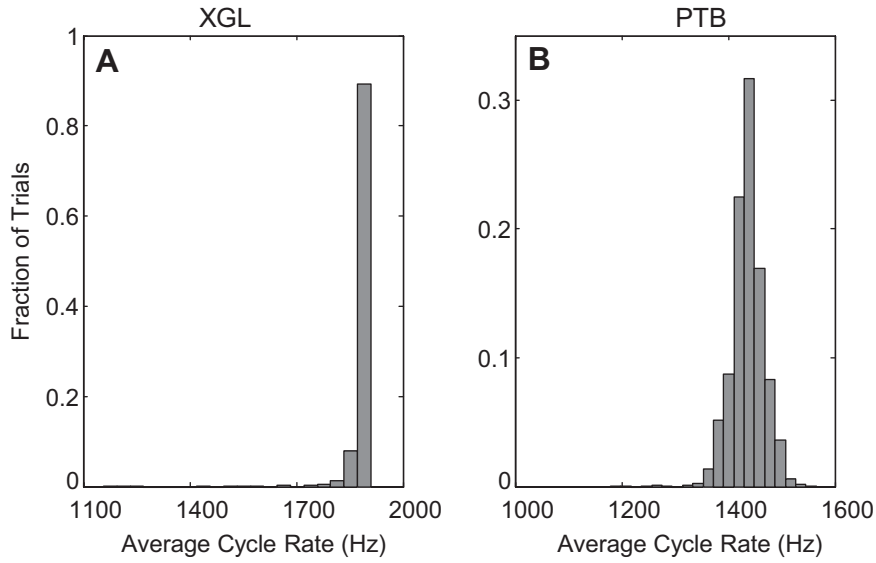

Fig. 8. Histogram of average cycle rates across all image trials. *A*: average cycle rate across 2,400 image trials using XGL was 1,883 Hz. The average cycle rate during the 1st trial was 1,898 Hz, more than double the 913 Hz reported for images in the previous report (Asaad and Eskandar 2008b). 99% Of trials had an average cycle rate >1,700 Hz. 0.04% Of trials had average cycle rates <1,200 Hz, and the minimum average cycle rate across all trials was 1,157.3 Hz. *B*: average cycle rate across 1,500 image trials using PTB was 1,435 Hz. The average cycle rate during the 1st trial was 1,436 Hz. 99% Of trials had an average cycle rate >1,350 Hz. 0.07% Of trials had average cycle rates <1,200 Hz, and the minimum average cycle rate across all trials was 1,179 Hz.

routine, during which data from the device are read and written to the behavioral data file. Previously, DAQ was halted as soon as the end_trial was called, and data were immediately read from the DAQ system using the "getdata" command. The number of samples requested was specified using the SamplesAvailable parameter. This had resulted in the lost samples at the end of the trial in previous versions of the software. In the current version, before stopping acquisition, a loop has been added that monitors the size of SamplesAvailable until it reaches the number of samples expected (typically requiring 1 update to SamplesAvailable) and then stops acquisition and reads the data from the device. This ensures that there will be data samples corresponding to the last few milliseconds of behavioral events that may have immediately preceded the function call to end the trial. Therefore, the trial exit time may increase to accommodate this requirement.

DISCUSSION

This report documents recent progress in the development of a comprehensive MATLAB-based software platform for presenting sensory stimuli and controlling behavioral tasks and describes its performance in a real-world experimental setting. Important new additions to the software include the implementation of a more flexible stimulus-presentation system (PTB), the ability to display dynamic stimuli (i.e., movies), performance and stability enhancements, and features that facilitate monitoring subjects' behavioral performance.

A key addition to the software is the ability to present movies as visual stimuli, which addresses a major limitation of the previous version of the software. The ability to display and precisely control dynamic stimuli is a key requirement for a wide range of behavioral and neurophysiological studies. Importantly, the display of movie frames is temporally precise and is achieved without a substantial sacrifice in temporal performance of other key functions such as monitoring behavioral events, updating the experimenter's control screen, digital inputs and outputs, or reward delivery. At the present time, a limitation is that only pregenerated movies (i.e., AVI movie files generated before running a task) can be displayed. A goal for future development is to add the ability to create and display movies in real-time and/or in the ITI.

Several changes since the previous report have yielded substantial increases in temporal performance of the system, which alleviate limitations that made the previous version unsuitable for some applications. Notably, an issue that led to a ~25-ms "blind" period at the beginning of each behavioral tracking episode that was documented in a previous report

Table 3. *Temporal performance of USB and PCI data acquisition devices*

Input/Output Type	Mean, ms	Standard Deviation, ms
putsample (USB)	1.3	0.15
putsample (PCI)	0.55	0.035
getsample (USB)	14.3	0.26
getsample (PCI)	0.66	0.17
putvalue (USB)	1.3	0.15
putvalue (PCI)	0.58	0.019
getvalue (USB)	1.3	0.12
getvalue (PCI)	0.55	0.035

Values reported represent average latency in milliseconds of various function calls to USB and PCI data acquisition devices.

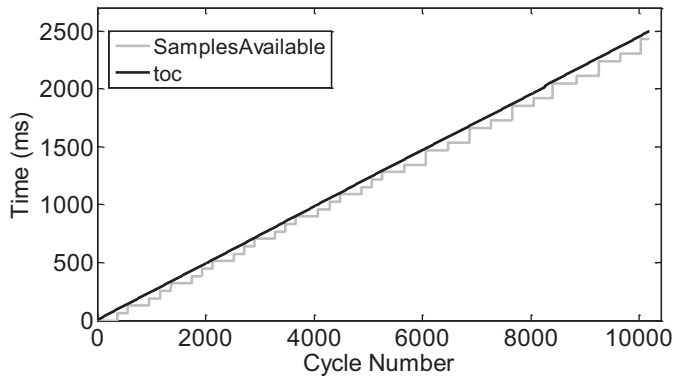

Fig. 9. Comparison between time stamps and number of samples acquired from a hardware device. The toc function (black) and SamplesAvailable (gray) are shown here. Toc is used to provide time stamps during data acquisition, and SamplesAvailable (a property of MATLAB Data Acquisition Toolbox) indicates the number of samples collected from the hardware device.

(Asaad and Eskandar 2008b) has been solved. We also demonstrate that the acquisition of analog signals through the Data Acquisition Toolbox in MATLAB can cause samples to be lost at the end of trials, typically <200 samples in a trial. We present a modification to the software to ensure that all samples acquired by the hardware are stored in the data file for further analysis. With several exceptions (as described in RESULTS), temporal performance of most functions and operations has improved due to both optimizations of the software and improvements in computing hardware, software, the operating system, and device drivers. Although the software already delivers adequate performance for most psychophysical and neurophysiological studies, certain approaches could make use of further improvements beyond that reported here. For example, temporally precise electrical or optical microstimulation might require microsecond-level temporal precision.

For researchers studying visual perception and visual neurophysiology, the ability to generate easily a wider range of visual stimuli (beyond simple predefined shapes, static images, and pregenerated movies) would be a valuable addition to the software. For example, the ability to implement transparency, dynamic stimuli that can be generated or controlled in real-time, and stereoscopic image presentation are all currently unsupported but would benefit the vision research community. The implementation of PTB (Brainard 1997; Kleiner et al. 2007; Pelli 1997) as the graphics system will provide a foundation for enhanced image and movie performance through future development.

A central goal and motivation for the development of this software is to provide a tool for the quick and easy creation of new experimental paradigms and behavioral tasks while maintaining tight control over stimulus presentation and behavioral events. The implementation of the software in MATLAB is a significant advantage because of the wide adoption of MATLAB as a tool for data analysis in our field. Similar systems that are implemented using a low-level programming language (e.g., C or C++) require nontrivial computer programming skills, which can be discouraging or even prohibitive for researchers that do not have a strong programming background. Furthermore, behavioral tasks coded in such programs tend to require more lines of code and have a much higher complexity than the same programs written in our software because of the greater abstraction it provides; this simplicity is beneficial for debugging and the avoidance of insidious errors in task execution. High-level programming environments such as MATLAB have proven to be easier to learn for many researchers and have thus enjoyed wide adoption. This has the added benefit that a single programming environment can be used for all stages of a project, including experiment design, behavioral training, data collection, and analysis.

MonkeyLogic is just one of several available software packages designed for behavioral control. It was created loosely based on the COmputerized Real-Time EXperiments (CORTEX; http://dally.nimh.nih.gov) software developed at the National Institute of Mental Health. Therefore, it shares broad features with CORTEX such as the use of a conditions file that explicitly enumerates the stimuli to be used in different trial types and a timing file that is called on to control the actual ongoing events (stimulus presentation and behavioral monitoring) within each trial. The software handles the transitions between and selection of trials (according to user-specified criteria) and automatically generates a standard behavioral data file. This is unlike the approach taken by several other software systems such as Lablib (http://maunsell.med.harvard.edu/software.html) or independently run PTB (Brainard 1997; Kleiner et al. 2007; Pelli 1997), in which the experimenter is responsible for explicitly coding all events, such as for the selection of trials and blocks, for saving behavioral data, etc. Those software packages are designed primarily as a tool that gives the researcher access to low-level video presentation and behavioral-signal-acquisition hardware rather than as a framework for trial-based experimentation.

Although such other software may, for some, provide a sense of freedom to design a task completely from scratch and unconstrained by the trial-based structure of systems such as MonkeyLogic, this structure does allow for some advantages. For example, three main features of our approach are: *1*) behavioral contingencies (e.g., wait for subject to fixate on a particular target regardless of where that target happens to be at any given moment) can be set using a few simple options within an abstract function call rather than through a millisecond-by-millisecond programming of each contingency as it occurs; *2*) trial randomization and block order often can be determined using straightforward graphic user interface (GUI) options; and *3*) standardized data files contain complete and easily accessible behavioral records (e.g., any trial performed in MonkeyLogic can be replayed to watch the subject's performance through a simple GUI).

Our software does not currently provide an option for drag-and-drop creation of behavioral tasks as is available in some other behavioral control software. Therefore, a basic familiarity with MATLAB programming is necessary to code tasks in a scripted rather than graphic fashion. Rather than develop such a GUI, we have instead focused our efforts thus far on maximizing the simplicity of the scripting requirements: for example, there is just one basic command for delivering stimuli and another for monitoring performance; these flexibly accommodate the modality of the stimulus being delivered (e.g., visual static, visual movie, auditory, electrical, etc.) and the type of behavioral signal being monitored (joystick, eye position, and digital or analog buttons). In our experience, given the relative simplicity of the MATLAB language and MonkeyLogic-specific functions, the requirement to code tasks manually has not been a significant barrier to adoption or use even for novice programmers.

Nevertheless, there are clearly certain types of experiments for which our software is not well-suited. For example, a free-viewing task in which subjects are simply required to watch a long video while gaze position is being tracked would be unfeasible in MonkeyLogic because, to maintain timing accuracy, our software loads all stimulus data to video or PCI-DAQ memory (for visual cues or analog stimulation, respectively) rather than streams from a more distant storage medium such as hard disk, optical drive, or even motherboard RAM. Likewise, experiments in which high-temporal determinism (e.g., electrical stimulation must be delivered to within submillisecond precision of another event) are likely to encounter significant barriers in MonkeyLogic as well as in any pseudo-real-time behavioral control system running in a high-level programming language such as MATLAB.

It is our hope that this software will improve productivity and lower the energy barrier for developing new experiments and testing more hypotheses. Additional information about the software can be found at http://www.monkeylogic.net.

ACKNOWLEDGMENTS

We thank Markus Siegel, Camillo Padoa-Schioppa, Chris Rishel, Tim Buschman, Valerie Yorgan, John Gale, Sam Barnett, Jillian McKee, Sruthi Swaminathan, Arup Sarma, and Arjun Venkataswamy for major contributions to the software, beta testing, and helpful discussions. We also thank Jeffrey S. Perry for making the low-level graphic drivers publicly available and for helpful advice regarding their implementation and the Psychtoolbox development team. We also thank the many users of the software that have provided bug reports, suggestions for new features, beta testing, and feedback. MATLAB is a registered trademark of The MathWorks, Inc.

GRANTS

Funding was provided by National Institute of Neurological Disorders and Stroke Grant R03-NS-067322, National Eye Institute Grant R01-EY-019041, and National Science Foundation CAREER Award 0955640.

DISCLOSURES

No conflicts of interest, financial or otherwise, are declared by the author(s).

AUTHOR CONTRIBUTIONS

W.F.A., N.S., S.M., and D.J.F. conception and design of research; W.F.A., N.S., S.M., and D.J.F. performed experiments; W.F.A., N.S., S.M., and D.J.F. interpreted results of experiments; W.F.A., N.S., and D.J.F. drafted manuscript; W.F.A., N.S., S.M., and D.J.F. edited and revised manuscript; W.F.A., N.S., S.M., and D.J.F. approved final version of manuscript; N.S. and D.J.F. analyzed data; N.S. and D.J.F. prepared figures.

REFERENCES

Asaad WF, Eskandar EN. Achieving behavioral control with millisecond resolution in a high-level programming environment. *J Neurosci Methods* 17: 235–240, 2008a.

Asaad WF, Eskandar EN. A flexible software tool for temporally-precise behavioral control in MATLAB. *J Neurosci Methods* 174: 245–248, 2008b.

Brainard DH. The Psychophysics Toolbox. *Spat Vis* 10: 433–436, 1997.

Gardner J, Larsson J. MGL Toolbox. In: *Gardner Research Unit | mgl: overview* (Online). http://gru.brain.riken.jp/doku.php/mgl/overview, 2012.

Kleiner M, Brainard D, Pelli D. What's new in Psychtoolbox-3? *Perception* 36, *ECVP Abstract Suppl*: 2007.

Lee D. Picto. In: *Lee Lab* (Online). http://leelab.yale.edu/Software.html [2012].

Maunsell JH. *Lablib* (Online). http://maunsell.med.harvard.edu/software.html [2012].

National Institute of Mental Health, Laboratory of Systems Neuroscience, Section on Neurophysiology. CORTEX. In: *Software and Hardware for Neurophysiology* (Online). http://dally.nimh.nih.gov [2012].

Neurobehavioral Systems. E-Prime. In: *Neurobehavioral Systems* (Online). http://www.neurobs.com [2012].

Pelli DG. The VideoToolbox software for visual psychophysics: transforming numbers into movies. *Spat Vis* 10: 437–442, 1997.

Perry JS. XGLToolbox. In: *Space Variant Imaging, Center for Perceptual Systems, University of Texas at Austin* (Online). http://fi.cvis.psy.utexas.edu/software.shtml, 2012.

Psychology Software Tools. Presentation. In: *Psychology Software Tools: Stimulus Presentation Software and Hardware for Research, Assessment, and Education* (Online). http://www.pstnet.com [2012].

Rose J, Otto T, Dittrich L. The Biopsychology-Toolbox: a free, open-source Matlab-toolbox for the control of behavioral experiments. *J Neurosci Methods* 175: 104–107, 2008.

Zeki S. Cogent 2000. In: *Cogent | Wellcome Laboratory of Neurobiology* (Online). http://www.vislab.ucl.ac.uk/cogent.php, 2012.