NIH Public Access

Author Manuscript

J Neurosci Methods. Author manuscript; available in PMC 2009 September 30.

Published in final edited form as:

J Neurosci Methods. 2008 September 30; 174(2): 245–258. doi:10.1016/j.jneumeth.2008.07.014.

A flexible software tool for temporally-precise behavioral control in Matlab

Wael F. Asaad* and
Department of Neurosurgery, Massachusetts General Hospital, Boston, MA, 02114, USA

Emad N. Eskandar
Program in Neuroscience, Harvard Medical School, Boston, MA, 02115, USA

Abstract

Systems and cognitive neuroscience depend on carefully designed and precisely implemented behavioral tasks to elicit the neural phenomena of interest. To facilitate this process, we have developed a software system that allows for the straightforward coding and temporally-reliable execution of these tasks in Matlab. We find that, in most cases, millisecond accuracy is attainable, and those instances in which it is not are usually related to predictable, programmed events. In this report, we describe the design of our system, benchmark its performance in a real-world setting, and describe some key features.

Keywords

Neurophysiology; Psychophysics; Matlab; Behavioral Control; Software; Cognition; Human; Monkey

INTRODUCTION

Interesting neural data are often the products of well-designed, psychophysically-rigorous behavioral paradigms. The creation and execution of these behavioral tasks relies upon a small range of applications that run on a relatively narrow range of software and hardware (Hays et al., 1982; White et al., 1989–2008; Ghose et al., 1995; Maunsell, 2008). The strengths and weakness of each application reflect the types of behaviors studied at the time of their initial development. Too often, the transition towards different types of behavioral tasks strains the flexibility of these programs, and cumbersome workarounds layer successively upon one another.

Recently, however, the performance of even a higher-level programming environment, specifically Matlab, has been demonstrated to be adequate for behavioral control at the one millisecond time-scale (Meyer and Constantinidis, 2005; Asaad and Eskandar, 2008). Thus, although no software running on Windows can attain truly deterministic, hard-real-time performance (Ramamritham et al., 1998), such software can nevertheless deliver high (not

*Corresponding Author, Address: Department of Neurosurgerym Edwards Building, Room 426m Massachusetts General Hospital, Boston, MA, 02114, E-mail: wfasaad@alum.mit.edu, Phone: 617-905-7691, Fax: 617-726-2310.

Publisher's Disclaimer: This is a PDF file of an unedited manuscript that has been accepted for publication. As a service to our customers we are providing this early version of the manuscript. The manuscript will undergo copyediting, typesetting, and review of the resulting proof before it is published in its final citable form. Please note that during the production process errors may be discovered which could affect the content, and all legal disclaimers that apply to the journal pertain.

NIH-PA Author Manuscript

NIH-PA Author Manuscript

NIH-PA Author Manuscript

NIH-PA Author Manuscript NIH-PA Author Manuscript NIH-PA Author Manuscript

perfect) temporal reliability. Given those data, we now focus on the design, real-world performance, and usability such a system can achieve.

In particular, we sought to harness the Matlab high-level programming environment to allow the quick and efficient coding behavioral tasks. By creating a system that has minimal programming overhead, we hoped to allow users to focus on the essential features of experimental design and the basic elements of behavioral control and monitoring rather than on the often arcane details of the video presentation and data acquisition hardware. Our major goals were:

- To allow behavioral control with high temporal precision in Matlab.
- To allow straightforward scripting of behavioral tasks using standard Matlab syntax and functions.
- To interface transparently with data acquisition hardware for input / output functions, such as eye-signal, joystick and button-press acquisition, reward delivery, digital event marker output, as well as analog and TTL output to drive stimulators and injectors.
- To allow the full reconstruction of task events from the behavioral data file by including complete descriptions of behavioral performance, the event markers and their text labels, the task structure, and the actual stimulus images used; as a demonstration of this goal, to allow the re-playing of any given trial from the behavioral data file alone.
- To provide the experimenter with an information-rich display of behavioral performance and to reflect task events in real-time to aid the assessment of ongoing behavior.

MATERIALS AND METHODS

Our tested system was composed of a Dell Computer with a Pentium Core 2 Duo processor (model 6300) running at 1.86 GHz and containing 1 GB of RAM (Dell Inc, Round Rock, TX). The operating system was Microsoft Windows XP, service pack 2 (Microsoft, Redmond, WA). The graphics hardware in this machine consisted of an nVidia Quadro NVS 285 with 256 MB of video RAM. Output from this dual-headed graphics card was split to two subject displays running in full-screen mode at pixel resolutions of 800 × 600, and an experimenter's control display, running in windowed mode at a resolution of 1024 × 768. The displays were standard cathode-ray tubes measuring 15 inches in the diagonal, also from Dell. The refresh rate for the tests reported here was 100 Hz, and video was double-buffered. The experimenter's display window was set to update every 100 ms during behavioral monitoring to allow near-real-time observation of the subject's performance.

Matlab software (version r2007b, The Mathworks Inc, Natick, MA), including the Data Acquisition Toolbox and the Image Processing Toolbox, was used to write the behavioral control software tested here, and to analyze the timing data reported below. All functions comprising our software were written as simple ".m" files that are directly editable by any user. Matlab was run in the default, non-multi-threaded mode. Matlab figures for the experimenter's display (created using the built-in graphics functions) relied upon OpenGL with hardware acceleration enabled. For the subject's display, low-level routines for video control (based on DirectX from Microsoft Corp.) were obtained through the generosity of Jeffrey S. Perry at the University of Texas at Austin. All tests were run within Matlab with the Java Virtual Machine disabled (launched by typing "matlab –nojvm" at the windows command prompt).

NIH-PA Author Manuscript

NIH-PA Author Manuscript

NIH-PA Author Manuscript

An optimized system profile was created as described previously (Asaad & Eskandar, 2008) to minimize the amount of processor time that could be stolen by other applications and devices. In addition, increasing the process priority of Matlab in Windows effectively decreased the amount of time stolen from the behavioral task by other applications (Asaad and Eskandar, 2008). Therefore, trials were run at the highest process priority allowed by Windows ("Real Time"), and the priority was lowered to "Normal" during the inter-trial-intervals to allow other pending activities time to execute. (Note that setting the process priority for trials is accessible to the user through an option in the main menu of our software.)

Behavioral signals were monitored using two identical data acquisition boards (a.k.a., DAQ boards) from National Instruments: two PCI-6229 multi-function DAQ cards were each connected to a BNC-2090a break-out box (National Instruments, Austin, TX). These were interfaced to Matlab using the Data Acquisition Toolbox. Although this toolbox is not intended for real-time control, our tests (Asaad and Eskandar, 2008) have suggested that it is nevertheless capable of delivering the performance necessary for millisecond-level behavioral monitoring.

We split the incoming behavioral signals into two analog input boards to allow more rapid sampling and simultaneous storage of these data. This is because logging and sampling data from the same board would be subject to upload delays caused by the temporary storage of samples in the acquisition board's local memory buffer (Asaad and Eskandar, 2008). An option in our software automatically detects the presence of two identical DAQ boards, and will allocate one for storage and one for on-line sampling.

Digital event-markers were written to a separate neural data acquisition system (Plexon, Dallas, TX) using the parallel port rather than the digital outputs on the DAQ card because we found that, on our system, the parallel ports were significantly faster (Asaad and Eskandar, 2008).

To assess the performance of our software, we analyzed data from the on-going training of a subject. Specifically, we employed a simple behavioral task in which a rhesus monkey (*macaca mulatta*, male, 6.1 Kg) was presented with four objects simultaneously, and needed to learn to pick one of the four at the end of a short delay by trial-and-error. To allow eye-tracking, head fixation was achieved using a head-post system (Judge et al., 1980). Visual fixation was required for a total of 2.5 seconds (1 second of initial fixation followed by a 500 ms cue presentation and then a 1 second delay) before the execution of a saccadic response to select a target. An inter-trial-interval of 2 seconds was used. Data from three consecutive days of training (one session each day) were collected and confirmed to yield nearly identical results, so one of these sessions was chosen arbitrarily for presentation below. This session consisted of 1601 trials over 2 hours and 53 minutes. At all times, the animal was handled in accord with NIH guidelines and those of the Massachusetts General Hospital Animal Care and Use Committee.

Analog X & Y position signals conveying behavioral output consisted of an optical eye-tracking system (Iscan, Inc., Burlington, MA) running at 120 Hz. Although the relatively slow speed of eye-tracking used here did not necessitate millisecond-level accuracy (unique behavioral samples were available only every 8.3 milliseconds), 1 ms accuracy is nevertheless the standard to which behavioral control systems such as ours aspire. A joystick (JC200 multi-axis inductive controller from Penny & Giles, Dorset, U.K.) was used to test the behavioral task during initial scripting.

For a more straightforward demonstration, a schematic diagram, the code and the conditions file for a simpler task (a standard delayed-match-to-sample, or DMS, task) is shown in Figure 4 (the task actually tested, described above, consisted of 96 conditions and more lines of code to assess on-line learning and handle block selection).

NIH-PA Author Manuscript

NIH-PA Author Manuscript

NIH-PA Author Manuscript

RESULTS

First we describe the basic design strategy of the software and the potential strengths and weaknesses of our approaches. We then examine the temporal performance of the software, specifically in the context of an actual behavioral task (rather than as in the more abstract tests described in Asaad & Eskandar, 2008). Finally we describe some features intended to enhance usability.

Design

The interactive structure of any behavioral task is defined by only two main activities: stimulus presentation and behavioral monitoring (corresponding to input and output, from the perspective of the subject). As such, our software is designed to facilitate these two activities by providing one function corresponding to each.

Stimulus presentation consists of the activation or inactivation of inputs to the subject that are intended to drive or constrain behavior and/or neural activity. Stimuli can be delivered through any of several modalities, including visual, auditory, electrical (analog output), or chemical (digital or analog output to an injector). These are specified by the experimenter in a table that lists the stimuli available for each trial type, or “condition” (Figure 3b), similar to the way that is done in CORTEX (White et al., 1989–2008), but with the direct specification of stimulus objects in this file rather than through an index into another “items” file (while this has the advantage of placing all the stimulus information in direct view, this table can appear overly-dense at times). A condition is defined by the collection of stimuli that are employed for a particular trial, or the contingencies that determine the timing and manner in which these stimuli are presented. Then, within any individual trial, the activation or inactivation of a stimulus is accomplished through a call to the stimulus presentation function, *toggle*, with arguments corresponding to the stimuli to be turned on or off.

While the activation or inactivation of a stimulus is treated like an instantaneous event (even though the stimulus itself may persist in time), behavioral monitoring, on the other hand, is an activity that explicitly specifies the passage of time. Repeated observations of variables reflecting a subject’s behavioral or neural output are processed until a certain condition is met or a specified amount of time has elapsed. These variables, in practice, generally consist of electrical signals corresponding to such things as eye- or joystick-position, button presses, etc. Formally, the task of behavioral monitoring can be subdivided into just two complementary activities: 1) waiting for a signal to enter a target range, or goal or 2) waiting for a signal to leave a target range, or goal. Within our software, these tasks are accomplished through a call to the monitoring function, *track*, with arguments corresponding to which of these two activities is required, the signal to be monitored, the target(s), the threshold around the target(s), and the time allowed to achieve the goal.

When tracking for target acquisition (i.e., waiting for the behavioral signal to enter a particular goal range), multiple stimuli can be specified as potential targets using standard Matlab vector notation. The *track* function can check the behavioral signal’s position against one or many targets very efficiently because of the vectorized nature of the Matlab language. The output of this function will be a scalar indicating which target was acquired, or zero if none were acquired.

A task is constructed simply by interleaving these two functions, *toggle* and *track*, within the requisite conditional structures relating a subject’s behavioral or neural output to the appropriate stimuli to be delivered. Concretely, the user must write a Matlab script that calls these functions at the appropriate times, and provide a table of conditions that indicates which stimuli are to be shown on which types of trials. The way these user-provided elements fit into the over-all program flow is shown in Figure 1.

NIH-PA Author Manuscript NIH-PA Author Manuscript NIH-PA Author Manuscript

Calling the *track* function invokes a loop that performs the necessary low-level functions to monitor the selected signal. Each cycle of the loop samples the behavioral signal, transforms this signal into calibrated coordinates, and compares these coordinates against the positions of the possible targets. All of this is done automatically, based upon the parameters specified initially by the user. This is in contrast to many existing behavioral control software packages that leave the programming of this loop partly or entirely up to the user. We prefer the former approach for a several reasons. First, the execution of this loop is a core feature of any behavioral task, so requiring each user to code it independently results in a great deal of duplicated effort. Second, because the loop is standardized across different tasks, the performance of a task can be estimated, with some degree of certainty, based on previous experiments run on that machine (for example, on our machine, all tasks tested produced mean cycle rates between 900 and 1050 Hz). Lastly, while a special-purpose loop coded by an individual user tailored to his or her specific requirements is likely to offer superior performance, in terms of cycles executed per second, this is not likely to be of any ultimate benefit in most cases, as the cycle rate of this general purpose loop is sufficiently high to monitor behavior at the one millisecond time scale (see Performance, below).

This strategy works well when relatively infrequent events punctuate behavioral tracking, such as the appearance and disappearance of stimuli at several hundred millisecond intervals. However, this approach becomes increasingly cumbersome as the temporal intervals between behavioral monitoring and stimulus presentations decrease. For example, presenting a movie while monitoring behavior requires interleaving these functions in a manner not easily afforded by a single, general-purpose *track* function. In addition, because there are entry or exit costs to the *track* and *toggle* functions (see Performance, below), rapidly alternating between them could produce unacceptable gaps in behavioral monitoring. Nevertheless, it may be possible to find ways to incorporate movie presentation into this function without sacrificing temporal performance (Markus Siegel, personal communication), though this has yet to be confirmed.

Although the *toggle* and *track* functions contribute the bulk of functionality for the creation of any behavioral task, other functions are provided to allow for things such as digital time-stamping of behaviorally-relevant events, marking trials as correct or incorrect, defining interactive “hot keys,” etc.

Because scripting is done within the Matlab environment, its rich syntax and function set is available for maximum flexibility. In particular, the vectorized nature of the Matlab language is ideal for the manipulation of trial-by-trial data that naturally fall into these structures. For example, suppose one wanted to follow behavioral performance to find instances of several consecutive correct responses as an indication of learning:

Let **e** be a row vector of behavioral errors, one for each trial (with 0 = correct)

Let **n** be the number of consecutive correct responses sought (where n > 0)

Then:

```
f = find([1 logical(e) 1]); % determine boundaries between correct & incorrect trials

t = min(find(diff(f) > n)); % find intervals of the required length, then take first instance
```

where **t** is the number of the first trial that begins a sequence of **n** correct trials. Alternatively, one could take a moving average over the last **k** trials to assess if performance has surpassed some threshold, **m** (where 0 < **m** < 1):

```
if length(e) >= k,
    r = sum(~e(end−k+1:end)) / k > m;
end
```

NIH-PA Author Manuscript

NIH-PA Author Manuscript

NIH-PA Author Manuscript

where **r** is one if the threshold performance has been met, or zero otherwise. If, in this calculation, one wanted to consider only a certain subset of conditions, say conditions 3 and 4, in only the current block (say, 2), a simple modification is all that is needed:

Let **c** be the vector of condition numbers

Let **b** be the vector of block numbers

Then:

```
esub = e((c == 3 | c == 4) & b == 2);
```

and then substitute **esub** for **e** in the example above.

Those are just a few simple examples of the ways Matlab's syntax simplifies these sorts of tasks that are common in the realm of interactive behavioral control. Because vectors corresponding to behavioral errors, condition and block numbers – and a variety of other behavioral data – are available to the user's script on-line, Matlab's vector notation can be applied for the efficient manipulation of these arrays to assess behavior and modify the on-going task accordingly.

Performance

General Performance—At the beginning of each trial, functions such as toggle and track are initialized with information regarding the memory addresses of the video buffers containing the current trial's stimuli, and with the latest DAQ assignments and calibrations. In addition, data logging is initiated on the DAQ that is set to acquire data to memory. These events took 0.11 milliseconds, at maximum, as shown in Table 1.

At the end of each trial, the logged analog data and a record of time-stamps is retrieved before returning to the main loop which is responsible for selecting the next trial during the inter-trial interval (I.T.I.). These events took on average 14.1 milliseconds (28.1 milliseconds maximum). The trial entry and exit times do not impact behavioral tracking (because the user's timing script controls events between these time points), and so may be considered part of the I.T.I. However, because these times are not currently subtracted from the user's desired I.T.I. time, the actual I.T.I. will be longer by the sum of the trial entry and exit times.

These data take into account all trials except the first one. As shown in Table 1, there is a significant first-trial cost upon exiting a trial, and when first calling any sub-functions (here, *toggle, track*, and *eventmarker*). This is despite initializing those top-level functions as described previously (Asaad and Eskandar, 2008). Therefore, for some applications, it may be necessary to disregard the first trial. If this is not possible (e.g., during a non-stationary task in which every stimulus presentation counts towards learning), appropriately placed time-stamps that mark events relative to the deterministic system clock can alleviate, to some extent, the magnitude of this problem (fortunately, the *eventmarker* subroutine is susceptible to relatively smaller first-trial costs, as shown in Table 1).

During each I.T.I., the subsequent block and condition to be run is selected either using built-in options (e.g., "random without replacement" and "repeat errors immediately") or by user-provided Matlab scripts. All the relevant stimuli for that trial are loaded into memory and then transferred to video RAM. In addition, figures on the experimenter's display are updated to reflect the statistics of behavioral performance (e.g., percent correct, reaction times, etc.). When assessed using our sample task (see Methods), these events took about 99.5 milliseconds on average (115.5 ms maximum, see Table 1). This time will vary, however, with the number and size of stimuli to be processed. Therefore, the scheduled I.T.I. time is varied conversely to this measured preparation time to keep the actual time constant near the user's desired value.

NIH-PA Author Manuscript
NIH-PA Author Manuscript
NIH-PA Author Manuscript

Video Performance—The ability of Matlab to accurately control and time-stamp video displays has been described previously (Meyer and Constantinidis, 2005; Asaad and Eskandar, 2008), and Matlab is employed for visual stimulus presentation by software widely used in the psychophysics community (Brainard, 1997; Pelli, 1997). The key feature of any such system is the ability to accurately mark the time of the screen refresh ("flip") in order to use this information to determine the time of appearance of a visual stimulus (based on its screen position). In our particular system, the standard deviation of the jitter of time-stamps (stored upon software detection of a vertical blank) relative to the actual appearance of a stimulus (measured using a photoresistor) was 0.3 milliseconds (Asaad and Eskandar, 2008). Therefore, in what follows, we use software detection of the vertical blank as a surrogate for the photoresistor.

Four steps are required to present a visual stimulus on a standard computer display. First, that stimulus must be loaded into the computer's memory (usually from disk). Then, the image data must be passed to a memory buffer created for that image on the video card itself. In our software, these first two steps are performed during the inter-trial-interval. Every trial type ("condition") can be associated with one or more possible stimuli, and once a condition is chosen, all of its stimuli are loaded into video RAM. Because our goal was to minimize the amount of time needed to present stimuli from within a trial itself, performing these steps during the I.T.I. was considered the best option. The potential disadvantage of this strategy is the relatively smaller amount of video memory compared to system memory; it is possible that a trial using many large images or several large movies (not all of which would necessarily be displayed in any one trial, but all must be available) could exhaust the available video memory. Fortunately, the amount of video memory on modern graphics cards (currently 128 to 512 MB, enough to hold at least several hundred medium-sized stimuli typical of psychophysics experiments) is sufficient in most cases.

Next, the particular stimuli to appear at a given time must be transferred from video RAM to a specialized area of video memory that serves as a screen buffer (an operation termed a "blit"). A memory pointer can then indicate which buffer is currently the active screen. Redirecting this pointer to a new buffer is described as "flipping" the display. We kept the "vertical sync" enabled so that this flip could occur during only the vertical blank interval, preventing tearing artifacts. These processes of blitting then flipping are both performed at the time a stimulus is to called up to appear (at the issuance of a *toggle* command).

In our software, these events are largely hidden from the user; he or she needs to be concerned only with selecting the image(s) to be displayed, defining the screen position of each in degrees of visual angle, and then toggling the image(s) on or off at the appropriate times. Because graphics hardware and the associated drivers are constantly changing, our system interfaces with the low-level graphics drivers via a single gateway function. At least two publicly-available software packages allow low-level control of video hardware from within Matlab (Brainard, 1997; Pelli, 1997; Perry, 2008). All calls to video hardware act through this single function to execute the basic steps of visual stimulus presentation. If future generations of video hardware and associated control software necessitate a different method of interaction, changes will be required in only this gateway routine (written as a directly editable Matlab function) to interface with the updated drivers, in this way minimizing the time and labor required.

The temporal performance of the *toggle* function itself is shown in Table 1. Note that the time from when this function is invoked by the user to when it is ready to flip (i.e., the function "entry time") was always less than 2.3 ms (except in the first trial). However, because the flip will occur only at the time of the next vertical blank, a variable delay ensues. In our case, running at 100 Hz, the delay to the flip was always between 0 and 10 ms, and never more, indicating there were no skipped frames.

NIH-PA Author Manuscript

NIH-PA Author Manuscript

NIH-PA Author Manuscript

By default, the toggle function updates the experimenter's display with rectangles reflecting the size and color of the visual stimuli before returning control to the user. This is a time-consuming event, taking 25.9 ms on average (30 ms maximum). The user can elect to skip this step, in which case this function's exit time averages 0.9 ms (1.8 ms maximum).

Behavioral Monitoring Performance—Behavioral monitoring most often involves sampling one or more analog signals, transforming those signals into calibrated coordinates (e.g., angle of gaze or screen position of a joystick cursor), comparing those coordinates against possible targets, and intermittently updating a representation of the behavioral signal on the experimenter's display. For most psychophysical applications, the ability to perform these steps about once every millisecond is required. So, at a first approximation, the speed of a system can be assessed by counting the number of cycles executed per second. We found mean cycle rates on our test system approached 1 kHz, thereby suggesting that behavioral signals can be monitored at about one per millisecond, on average (Figure 2). Furthermore, sampling intervals were generally regular; specifically, delays greater than 2 milliseconds were rarely observed (99.9% of cycles were under 2 ms). Previously, we identified a temporal cost associated with the first cycle during which a screen update was requested (Asaad and Eskandar, 2008). Here, that update is called on the first cycle of each tracking epoch. Data from this task confirms that cycle times greater than 2.3 ms were observed exclusively on the first cycle within each call to the *track* routine. Within this first cycle, the highest recorded latency was 26.7 ms.

The cycle times for a typical tracking epoch are shown in figure 3a. The distribution of individual cycle times across all trials is shown in figure 3b (only the first cycle during each *track* call is excluded). There is a multi-modal distribution of cycle times where the highest modes (those above 1.5 ms) corresponded to cycles in which the position of a behavioral trace (a simple dot) on the control screen was updated (in our case, this occurred about once every 100 ms).

Yet, simply because increased cycle times were rare, it is possible that these episodes of increased latency were occasionally grouped in close temporal proximity (as could be due to a burst of high operating system activity, or to on-going background activity related to the plotting of the behavior trace), such that there were sporadic periods of unacceptably infrequent behavioral sampling. To evaluate this possibility, we examined the data from 1600 trials of our behavioral task (all but the first) and noted all cycle latencies greater than 1.5 milliseconds (i.e., those cycles in the higher mode of Figure 3b). The shortest interval between such high-latency events was 81 cycles. Another way of representing this data is shown in Figure 3c. Here, a histogram of cycle latencies is plotted relative to every high-latency event (at time zero). There is only a small tendency for latencies in the immediately following cycle to be increased (11.6 percent increase above the following bins or 0.11 ms in absolute time).

A more parametric way of showing this, at least for adjacent cycles, is depicted in Figure 3d. Here, no threshold was applied to the data. Rather, each cycle time is plotted against the subsequent one, yielding two interesting observations. First, the slope of the horizontally-oriented points was about 12%, equivalent to the result in Figure 3b, and the slope of the vertically-oriented points was about 2% (relative to the vertical), showing that there is indeed a very slight tendency for increased cycle times in those cycles preceding a high-latency event. Second, multiple modes are clearly visible. The cluster that appears between 1.2 and 1.3 ms (labeled as mode 2) consists nearly entirely of points corresponding to the second cycle within each tracking period. The lowest mode (below 1.2 ms) contained 99.0% of all points (because of a ceiling density effect in this figure, the relative magnitudes of these populations are more clearly appreciated in the logarithmic plot of Figure 3a).

NIH-PA Author Manuscript

NIH-PA Author Manuscript

NIH-PA Author Manuscript

These data confirm that behavioral tracking is generally very regular with a period of about 1 ms on our tested system, and cycles with increased latency occur at predictable times with respect to programmed events. Importantly, over all 1641 trials lasting nearly 3 hours, there was not a single non-initial cycle time greater than 2.3 ms.

Usability

Task Scripting—Figure 4 shows the elements necessary for constructing a simple delayed-match-to-sample (DMS) task. This task requires the subject to maintain fixation throughout an initial fixation period, a subsequent cue period, and finally a brief delay. Two pictures are then presented simultaneously and the subject must pick the one corresponding to the cue presented earlier. The conditions table (Figure 4b) shows 8 conditions comprising two pairs of stimuli (A & B or C & D). The timing script (Figure 4c) determines the times at which stimuli appear in relation to the subject's behavior; it consists of 43 lines of code (excluding the variable definitions). Of these, 28 are involved in simply saving an error code and aborting the trial if the subject fails to perform the requisite steps properly (e.g., breaking fixation). Therefore, it is possible that even more efficient scripting could be achieved by incorporating error handling as an option within the *track* function itself (though this would likely come at the expense of a more complicated *track* syntax).

Interfacing with I/O devices—Using the Matlab Data Acquisition Toolbox as a foundation, we constructed a system whereby a set of behaviorally-relevant signals can be mapped directly to specific inputs or outputs. In other words, analog input signals representing eye or joystick position, digital inputs representing button presses, digital or analog outputs for reward delivery, and digital outputs for sending event-markers to a separate neural data acquisition system, as well as other types of signals, can be assigned to specific channels (if analog) or lines (if digital) using a straightforward GUI. For example, to assign a signal to an input our output, simply select that signal (e.g., "Eye X Position"), then select the board (e.g., National Instruments PCI-6229), then the subsystem (e.g., "Analog Input"), and finally the channel (e.g., "1"), before clicking "Assign." Thus, any hardware input or output recognized by the Data Acquisition Toolbox can be used in the same manner. However, no explicit support is provided for serial ports, although these objects can be created and monitored by the user.

This approach allows the same task to be reconfigured easily to run on separate machines that may have different hardware connections, or to run on different types of hardware altogether, so long as the relevant channels or lines are visible to Matlab.

A potential disadvantage of this approach is that the types of behavioral signals that can be acquired are hard-coded into the system. In other words, while interfacing with two-dimensional signals such as eye-position and joystick-position is straightforward, and one-dimensional inputs such as buttons and levers are also recognized, more complicated signals that involve more than two dimensions of input are not directly supported (e.g., a signal representing multiple variables such as joint position or location in 3-D space). A user would need to create directly the Matlab data acquisition objects and code a behavioral monitoring loop to track these sorts of behavioral variables. Similarly, there is currently no support for monitoring digital inputs (such as might be used to direct the course of the behavioral task based upon input from a separate computer system dedicated to monitoring some aspect of behavioral output); this would also require a user-coded subroutine.

Because the polarity of the read (or "strobe") trigger bit on some neural data acquisition systems is reversed relative to that of parallel ports (e.g., the Neuroport system from Cyberkinetics), there is a menu option to invert this bit. Likewise, because some reward systems use a falling rather than a rising voltage to deliver reward, the polarity of this function can be reversed through a menu option as well.

NIH-PA Author Manuscript NIH-PA Author Manuscript NIH-PA Author Manuscript

Currently, analog data is not acquired into our system continuously. This is because acquiring continuous analog signals for the duration of an experiment (typically many minutes to several hours) would require intermittent transfer of that data to disk, possibly resulting in delays at inopportune moments during the behavioral task. Instead, analog data acquired to memory during each trial is retrieved at the end of that trial and saved to disk during the I.T.I. A major disadvantage of this approach is the loss of the analog data record during the I.T.I. Therefore, to perform analyses on eye-position during this interval, for example, one would need to split the analog signal into a separate neural data recording system that is capable of continuous analog data acquisition, and then use digital event-markers to align this signal with behavioral events.

Signal calibration—Behavioral signals such as eye or joystick position can be used in either a raw (i.e., pre-calibrated) manner, or they can be calibrated from within the software. This is done by presenting dots in sequence and marking the instant of fixation or joystick acquisition with a key-press. In contrast to some other calibration methods that take into account only signal offset and gain, our calibration method also takes into account skew using a projective transform. This greatly improves the quality of the calibration, especially when multiple, closely-spaced targets are used.

Most eye-tracking systems are subject to some degree of drift over time, even with measures such as tracking the pupil relative to the corneal reflection (on optical systems). Often, this drift is the result of some slight degree of head-movement during the task session. To counteract this, we employed a drift-correction algorithm (similar to what is available in other systems, such as CORTEX). Specifically, at the completion of each trial, fixations are extracted from the continuous eye-position record, and these fixations are compared to the position of targets that had been displayed; small errors are then assumed to reflect intended fixation on the center of those targets, and a fraction of this error is corrected. Using this method, we find that no manual intervention is needed, even over several hours, to keep the calibration exact.

Data file record—In our experience, data files generated by most behavioral control systems contain cryptic references to the conditions run, the stimuli shown, and the timing of key events. For example, they may contain unreferenced numeric markers that identify the trial type (condition), the code numbers and time-stamps of critical behavioral events, as well as a record of any acquired analog data. These numbers are then referenced to tables stored in separate files or notebooks that allow the reconstruction of trial events. Because these data are separate, it is possible that they can be lost or mis-associated, rendering the events undecipherable. Many researchers in this field have had the troubling experience of attempting to reconstruct the events of an experiment performed years ago by a colleague now long gone.

To remedy this, we included in our data files the fullest account of the behavioral record we thought possible and practical. These data files contain the actual stimulus images used, the text descriptions of the event markers, the conditions-table structure for the entire experiment, and a variety of task configuration information (e.g., screen resolution, signal calibration matrices, etc). This rich record allows the reconstruction and replaying of any given trial from this single data file, so there is no ambiguity about what events actually took place.

Because of their relative completeness, these data files are, unsurprisingly, somewhat large; a 3-hour session can generate a 50 MB file easily. In the long-term, however, we believe such file sizes will become only more practical as the power of typical PCs increases.

Two provided functions plot behavioral data in a graphical form to allow a quick over-view of events. The first plots the performance over trials and shows the reaction time histogram (Figure 5a). In addition, trials can be re-played, and movies of those trials created, from this figure

NIH-PA Author Manuscript

NIH-PA Author Manuscript

NIH-PA Author Manuscript

window. The second is called up from the first, and shows a timeline for the currently selected trial (Figure 5b).

Experimenter's display—At a minimum, during the execution of a behavioral task, most users will want some sort of real-time feedback which reflects the on-going behavior of the subject. For example, in a task in which eye-position (i.e., angle of gaze) is the monitored behavioral parameter, a moving point corresponding to the instantaneous gaze position will allow an observer to follow behavior to determine the reasons for a subject's successes or failures. Also, the experimenter will want to know this position relative to any objects currently visible on the subject's screen. Therefore, we constructed a display window that included markers corresponding to the current position of analog inputs such as eye or joystick position in relation to visual stimuli currently visible to the subject (Figure 6). In addition, objects reflecting lever position or digital inputs such as button position are visible if those inputs are active, and rings reflecting potential targets with their threshold radii also appear at the appropriate times.

This main display is surrounded by text indicating the current trial, condition, and block, graphs showing the performance within each of these groups, a reaction-time histogram, and a time-line of the event-markers stored for the previous trial.

We found that the time required to update the experimenter's display was linearly related to the number of graphical elements drawn in this figure window. The more polygons present, the longer the time that was required to update the display, even if the only change involved repositioning a small dot. Therefore, it is likely that faster performance (shorter cycle times) could have been achieved by simplifying this display. However, we chose to balance performance and usability. Specifically, we calibrated the complexity of our display to maintain cycle times of about 2 ms on our test machine on those cycles when updates were requested.

Configuration menu—A single user interface (Figure 7) is the starting point for loading an experiment, setting and saving a variety of configuration parameters (e.g., video settings, I/O mappings), and for setting the manner in which blocks and conditions are to be selected and what to do if the subject makes an error. While simple options such as "select conditions randomly with replacement," "choose blocks in increasing numerical order" or "immediately repeat incorrect trials" are explicitly available in this menu, Matlab functions can be used in place of these options to execute more complicated logic, such as detecting when a subject has learned and only then switching blocks, or always selecting certain conditions after others, or arranging the order of blocks in some specific manner, etc. Once a configuration has been created and saved for a particular behavioral task, launching an experiment involves loading the conditions file (which then automatically loads the associated configuration file and timing script(s)), entering a data file name, and clicking "Run."

Troubleshooting—To aid set-up, several diagnostic functions can be called from the main menu (Figure 7). Video tests are available to assess currently-selected video display options (e.g., resolution and refresh rate) and stimulus appearance at those settings. I/O tests are available to assess current DAQ assignments, to acquire or deliver individual analog or digital signals, and to test each digital line used for event-marker output. In addition, the maximum achievable on-line sampling rate can be estimated for the type and number of DAQs present. For optimizing on-line drift correction, an option exists to collect 3 seconds of eye data and show where saccades and fixations are detected according to the user's current settings. Lastly, one can collect a few hundred thousand latencies corresponding to open-loop cycle times to assess, grossly, Matlab's performance in the context of current system settings. In our experience, these address some of the more common problems to arise when first building a behavioral control system.

NIH-PA Author Manuscript NIH-PA Author Manuscript NIH-PA Author Manuscript

DISCUSSION

Being unsatisfied with currently available options for behavioral control, we sought to develop a high-level software system that simplifies task design and execution while maintaining a high degree of temporal precision. Matlab turned out to be an excellent platform for this project, as the timing constraints could be met while providing the user access to the simplicity and flexibility of that environment. Nevertheless, there are notable limitations.

Windows XP cannot support hard real-time operation. Therefore, while sub-millisecond jitter is acceptable in many, if not most, psychophysical settings, there are nonetheless many potential applications for which the software described here would not be suitable. In particular, experiments that must provide feedback within a very small temporal window (for example, to influence an on-going synaptic event) would find 1–2 ms jitter simply too variable. Likewise, delivering feedback that requires a great deal of processing could potentially incur unacceptably long delays unless these computations are programmed in a lower-level language.

There is a ~25 ms “blind” period at the beginning of each behavioral tracking episode. If a subject were to respond within that interval, it would not be appreciated until the end of this period. Therefore, in tasks in which behavioral responses are expected to occur very early in each tracking epoch and must be measured precisely, this software as it is currently designed would not be adequate. It would be possible to disable experimenter’s display updates, but that would significantly hinder one’s ability to follow behavioral events in real-time.

Other limitations include the current inability to display movies or translating visual stimuli while simultaneously tracking behavioral signals. In addition, behavioral signals are not currently stored during the inter-trial interval. The ability to store analog signals continuously would benefit not only behavioral signals, but neurophysiological ones as well. In other words, although many acquisition cards are clearly capable – in terms of number of channels, sampling rates and PC storage – of recording neural data alongside behavioral signals, no support has been built-in for this purpose. Fortunately, most users so far have preferred relying on a separate neural data acquisition system (e.g., Plexon). Nonetheless, such a capability would likely be useful for many potential applications.

We use this software on several different machines dedicated to neurophysiology in humans or non-human primates. This software has been very adept at the creation of basic sensori-motor tasks, and is especially useful for the creation of cognitive tasks with greater numbers of stimuli and contingencies. These tasks are often coded within an hour, and modifications are simple to test.

As with any endeavor, abstraction layers hiding lower-level details have certain benefits and potential pitfalls. For example, while such abstraction can improve ease-of-use and encourage adoption and innovation, it may also isolate one from those technical details that are critical to the task at hand; this could result misapplication or a false sense of limitation (ultimately, computers are capable of much more than any particular software system allows). Because our software constitutes a highly-abstracted environment, we hope that the benefits outweigh these costs. We hope that the lower the barrier to designing and executing behavioral paradigms, the more likely it is that one will explore the space of possible variations and implementations.

For lack of imagination and in the absence of a clever acronym, we refer to our software as “MonkeyLogic.” The software is available by request to the authors.

NIH-PA Author Manuscript

NIH-PA Author Manuscript

NIH-PA Author Manuscript

ACKNOWLEDGEMENTS

The authors thank David Freedman, Tim Buschman, Camillo Padoa-Schioppa, Valerie Yorgan, Markus Siegel, and John Gale for contributions to the software, beta testing, and helpful discussions. We also thank Jeffrey Perry for making the low-level graphics drivers publicly available and for helpful advice regarding their implementation. Anne-Marie Amacher, Ming Cheng, Jason Gerrard, Rollin Hu, Earl Miller, Andrew Mitz and Ziv Williams are appreciated for offering useful ideas for program testing and execution. We are indebted to the CORTEX development team (http://www.cortex.salk.edu) for their widely used behavioral control software, which provided an initial framework for the creation of our software. Funding was provided by a Tosteson Fellowship from the Massachusetts Biomedical Research Council to WFA and NEI grant 1R01DA026297, NSF IOB 0645886 and the HHMI to ENE.

REFERENCES

Asaad WF, Eskandar EN. Achieving behavioral control with millisecond resolution in a high-level programming environment. Journal of Neuroscience Methods. 2008

Brainard DH. The Psychophysics Toolbox. Spat Vis 1997;10:433–436. [PubMed: 9176952]

Ghose GM, Ohzawa I, Freeman RD. A flexible PC-based physiological monitor for animal experiments. J Neurosci Methods 1995;62:7–13. [PubMed: 8750079]

Hays, AV.; Richmond, BJ.; Optican, LM. A UNIX-based multiple-process system for real-time data acquisition and control; WESCON Conference Proceedings; 1982. p. 1-10.

Judge SJ, Wurtz RH, Richmond BJ. Vision during saccadic eye movements. I. Visual interactions in striate cortex. Journal of Neurophysiology 1980;43:1133–1155. [PubMed: 6766996]

Maunsell, JHR. LabLib. 2008. http://maunsell.med.harvard.edu/software.html.

Meyer T, Constantinidis C. A software solution for the control of visual behavioral experimentation. J Neurosci Methods 2005;142:27–34. [PubMed: 15652614]

Pelli DG. The VideoToolbox software for visual psychophysics: transforming numbers into movies. Spat Vis 1997;10:437–442. [PubMed: 9176953]

Perry, JS. XGL Toolbox. 2008. http://fi.cvis.psy.utexas.edu/software.shtml.

Ramamritham K, Shen C, Sen S, Shirgurkar S. Using Windows NT for Real-Time Applications: Experimental Observations and Recommendations. IEEE Real Time Technology and Applications Symposium. 1998

White, TM.; Norden-Krichmar, TM.; Benson, J.; Boulden, E.; Macknik, S.; Mitz, A.; Mazer, J.; Miller, EK.; Bertini, G.; Desimone, R. COmputerized Real-Time Experiments (CORTEX). 1989–2008. http://www.cortex.salk.edu/.

NIH-PA Author Manuscript

NIH-PA Author Manuscript

NIH-PA Author Manuscript

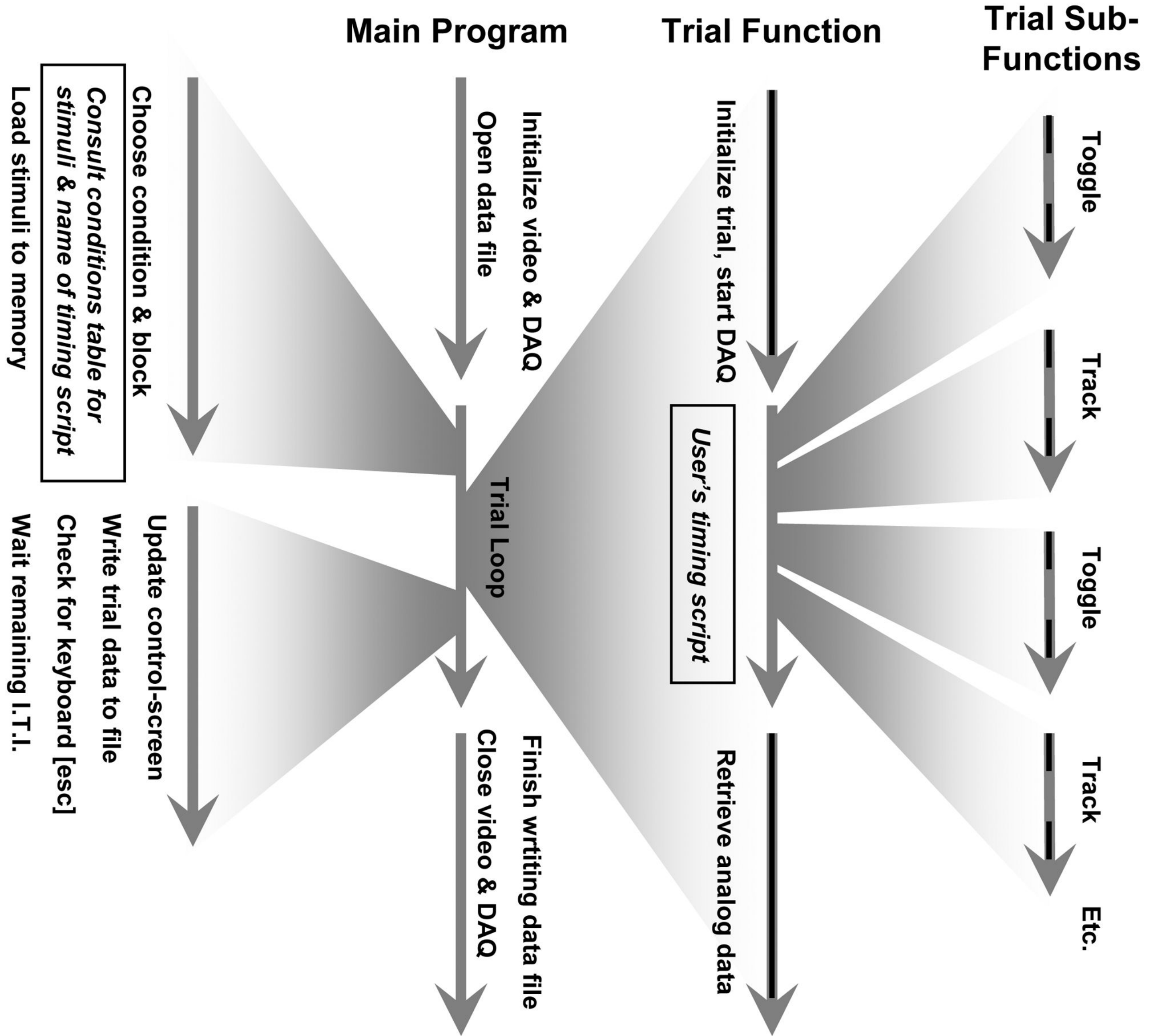


Figure 1. Program Schematic

The minimum elements that must be provided by the user are marked with boxes. In addition, user's can write Matlab scripts to control the time at which blocks change, the selection of blocks, and the selection of conditions. The darker portions of the arrows correspond to the "entry" and "exit" times measured in Table 1 (the *eventmarker* function is not shown, but would appear intermixed with *toggle* and *track*).

NIH-PA Author Manuscript NIH-PA Author Manuscript NIH-PA Author Manuscript

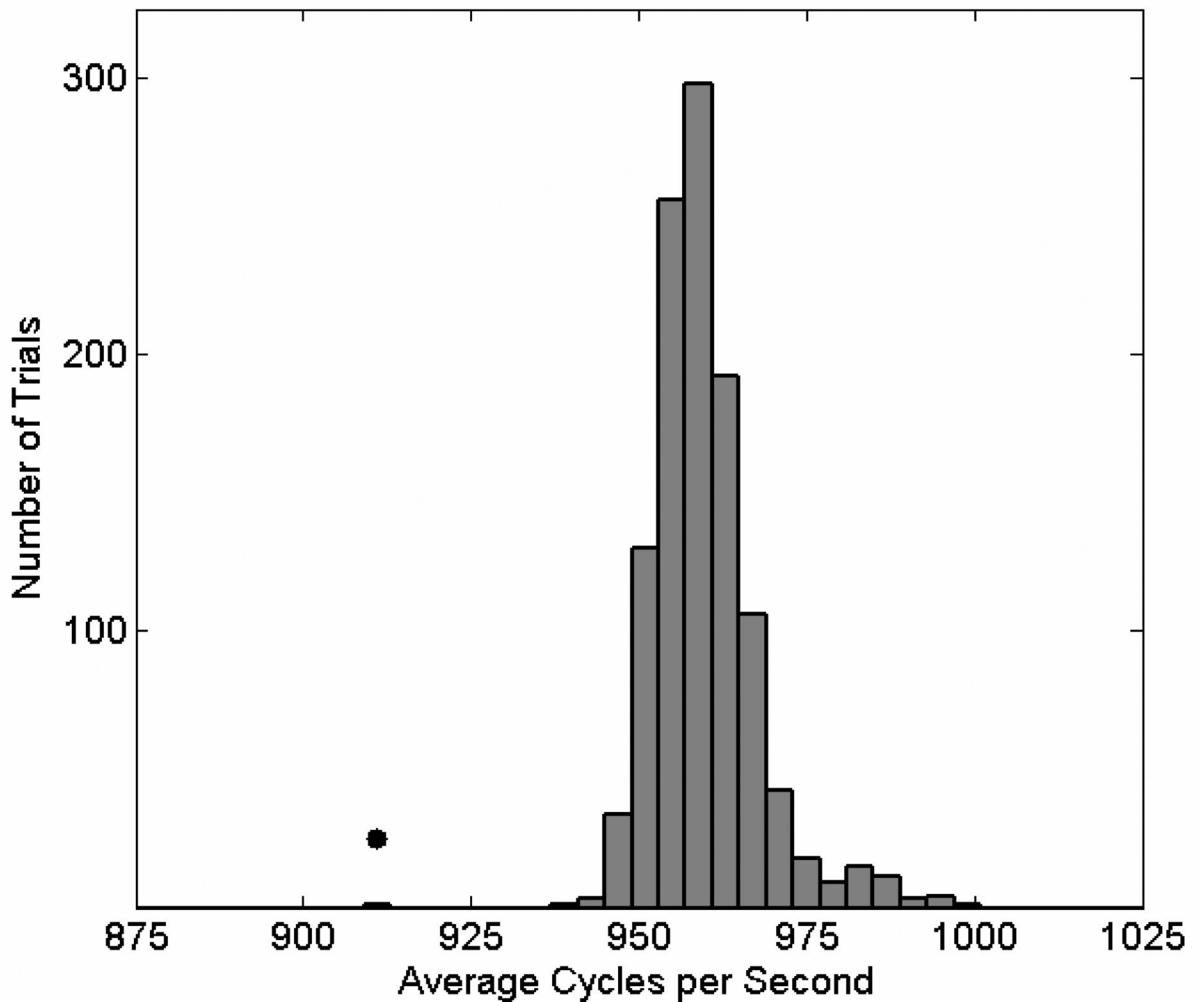
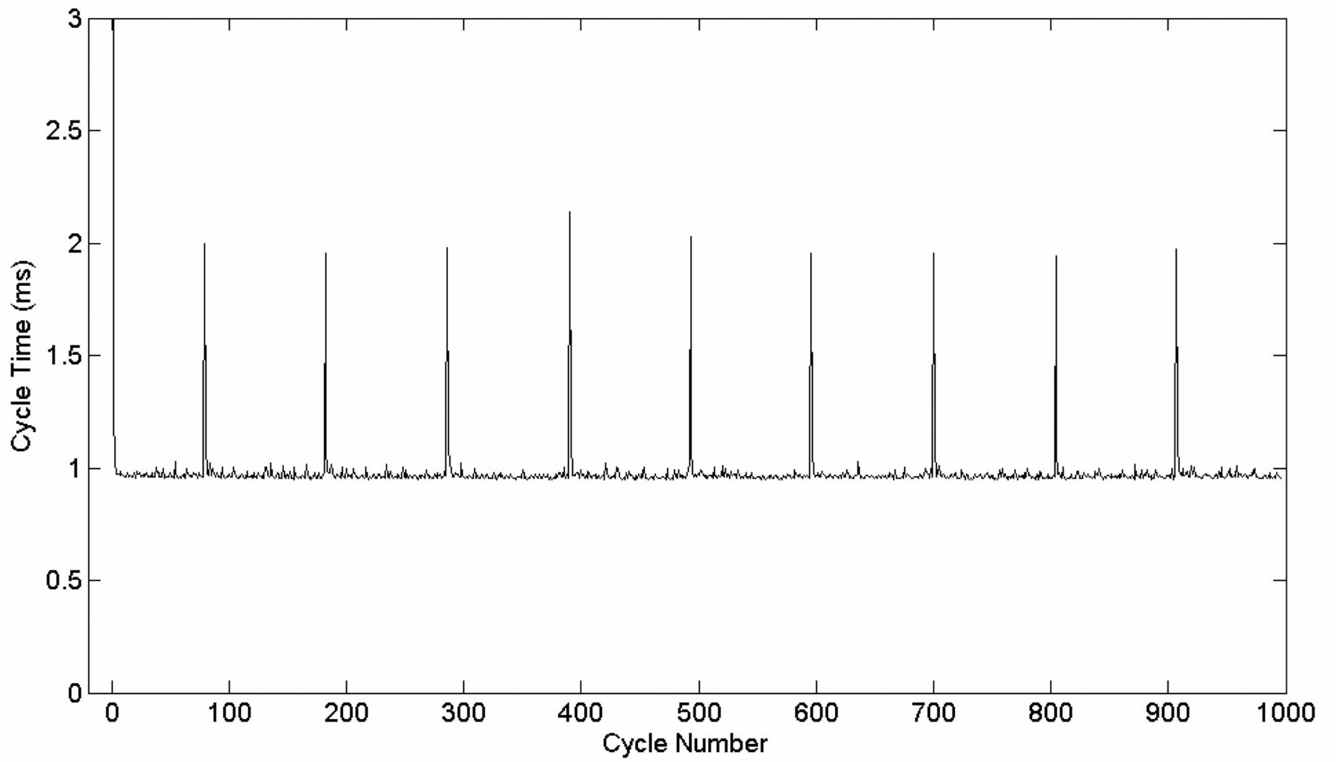


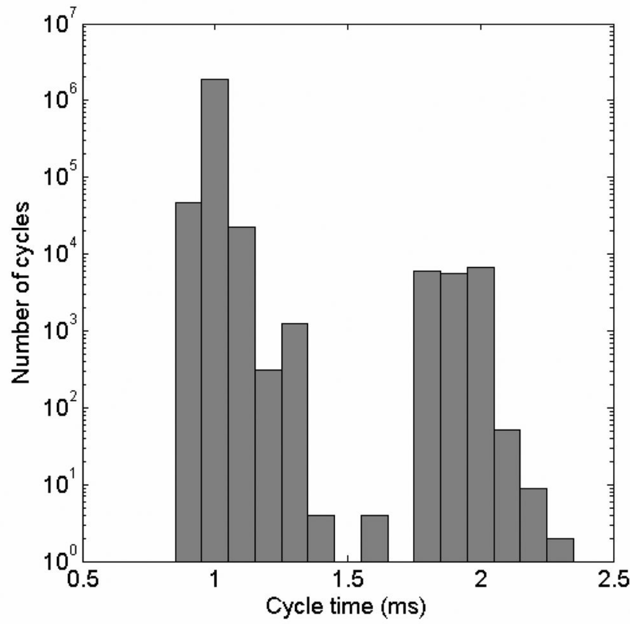
Figure 2. Measured Cycle-Rates

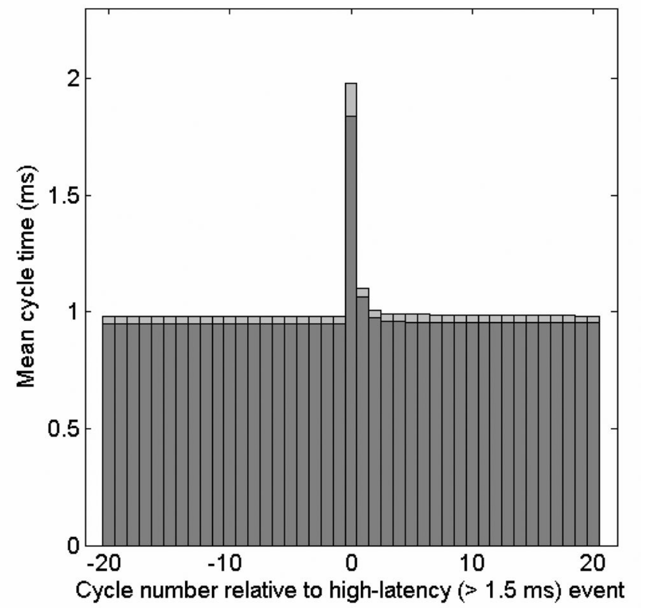
This histogram shows the mean cycle-rates for each trial over the behavioral session. As previously described (Asaad and Eskandar, 2008) the cycle rate of the first trial is typically lower, as marked here with a dot, representing 913 cycles per second. The next-slowest trial averaged 941 cycles per second. The mean cycle rate across all these trials except the first was 960. These rates include all cycles performed within the *track* function, including the first. Significantly faster cycle rates (> 2000 per second) have been observed on newer-generation PC systems.

Note that only correct-choice and incorrect-choice trials are shown, because other trial types placed different demands on stimulus presentation and tracking (i.e., average cycle-rates for break-fixation trials tended to be slightly lower, as relatively less time was spent tracking than was spent updating the screen, and no-fixation trials were slightly faster for the opposite reason). Over all trials, the range of cycle rates varied from 800 to 1020 Hz except for 3 instances in which the subject broke fixation nearly instantaneously, resulting in average cycle rates between 700 and 800 Hz.

NIH-PA Author Manuscript NIH-PA Author Manuscript NIH-PA Author Manuscript







NIH-PA Author Manuscript

NIH-PA Author Manuscript

NIH-PA Author Manuscript

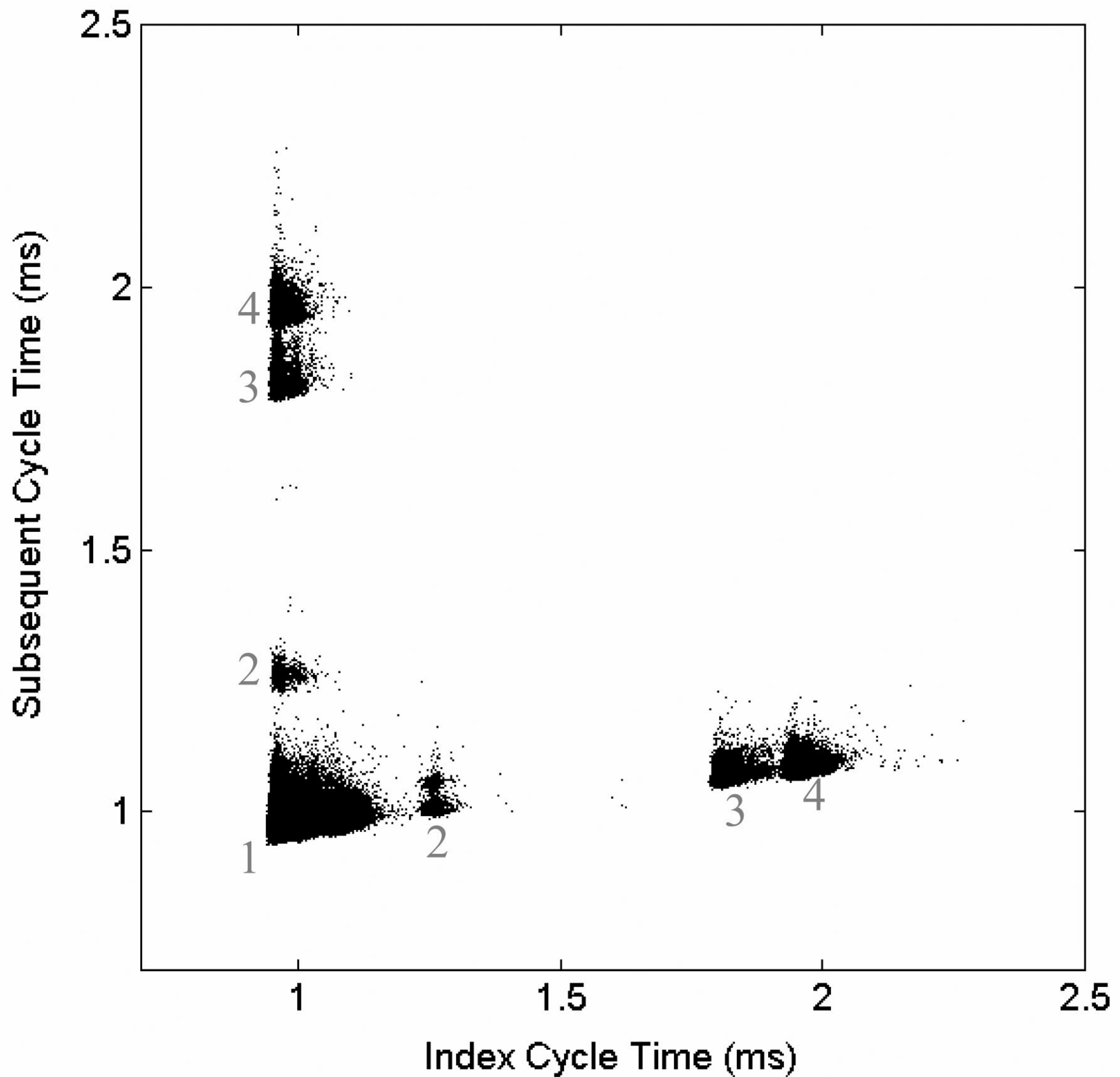


Figure 3. Behavioral Tracking Performance

(a) The cycle times for a typical behavioral tracking epoch are plotted. The y-axis is truncated at 3 ms (the first cycle time here is 26.5 ms). Note the periodically increased times corresponding to the 100 ms interval between updates to the behavioral trace in the experimenter's display. (b) The distribution of individual cycle times across all epochs of behavioral tracking in 1600 trials (the entire first trial, and the first cycle in each tracking epoch of all subsequent trials, were excluded). Cycle time is plotted against the number of cycles on a logarithmic scale. Cycle times in the higher mode (above 1.5 milliseconds) were found to correspond exclusively to those cycles during which the eye-trace on the experimenter's display was updated. (c) The relative distribution of high-latency events during behavioral tracking is shown. This histogram was triggered on the occurrence of behavioral tracking cycle times greater than 1.5 ms (here at time 0). The lighter shaded region at the top of each bar represents the area of the mean value +/− the standard deviation. The cycles immediately following the high-latency instances tended to be slightly increased in time (increased relative to baseline by 11.6%, or 0.11 ms). The minimum interval in any trial between two high latencies, each greater than 1.5 ms, was found to be 81 cycles. (d) A scatter diagram plotting

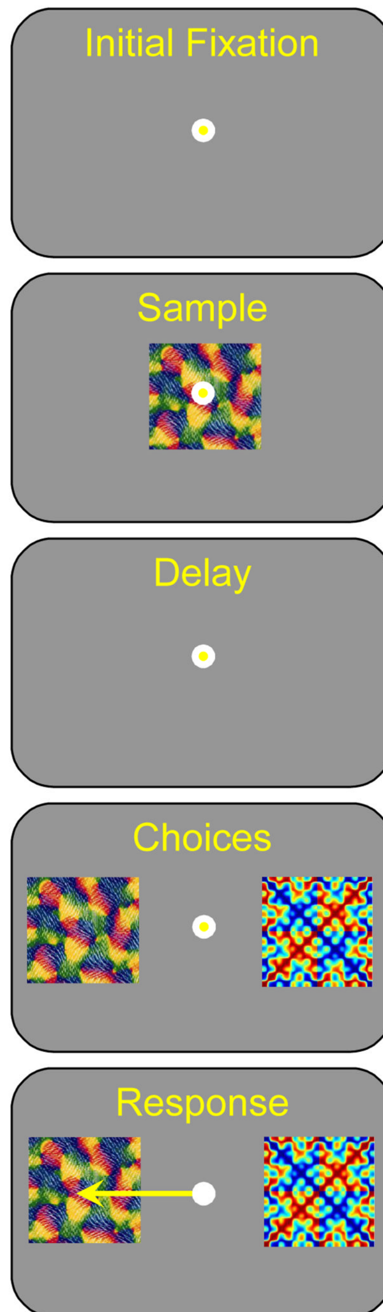
NIH-PA Author Manuscript

NIH-PA Author Manuscript

NIH-PA Author Manuscript

the time for each cycle against the time for the subsequent one (excluding the first cycle in each tracking epoch). Multiple modes are more clearly visible in this plot, but the relative numbers within each cluster are more difficult to ascertain than in 3a because of density saturation. Mode 1 contained 99.0% of all points, corresponding to a typical tracking cycle. Mode 2 corresponded to the second cycle within each tracking period. Modes 3 and 4 corresponded to those cycles in which a screen update request was made. No clear pattern of occurrence distinguished these last two modes.

NIH-PA Author Manuscript NIH-PA Author Manuscript NIH-PA Author Manuscript



NIH-PA Author Manuscript

NIH-PA Author Manuscript

NIH-PA Author Manuscript

Condition #	Relative Frequency	Condition in Blocks	Timing File	Object #1	Object #2	Object #3	Object #4
1	1	1 3	dms.m	fix(0,0)	pic(A,0,0)	pic(A,-5,0)	pic(B,5,0)
2	1	1 3	dms.m	fix(0,0)	pic(A,0,0)	pic(A,5,0)	pic(B,-5,0)
3	1	1 3	dms.m	fix(0,0)	pic(B,0,0)	pic(B,-5,0)	pic(A,5,0)
4	1	1 3	dms.m	fix(0,0)	pic(B,0,0)	pic(B,5,0)	pic(A,-5,0)
5	1	2 3	dms.m	fix(0,0)	pic(C,0,0)	pic(C,-5,0)	pic(D,5,0)
6	1	2 3	dms.m	fix(0,0)	pic(C,0,0)	pic(C,5,0)	pic(D,-5,0)
7	1	2 3	dms.m	fix(0,0)	pic(C,0,0)	pic(D,-5,0)	pic(C,5,0)
8	1	2 3	dms.m	fix(0,0)	pic(C,0,0)	pic(D,5,0)	pic(C,-5,0)

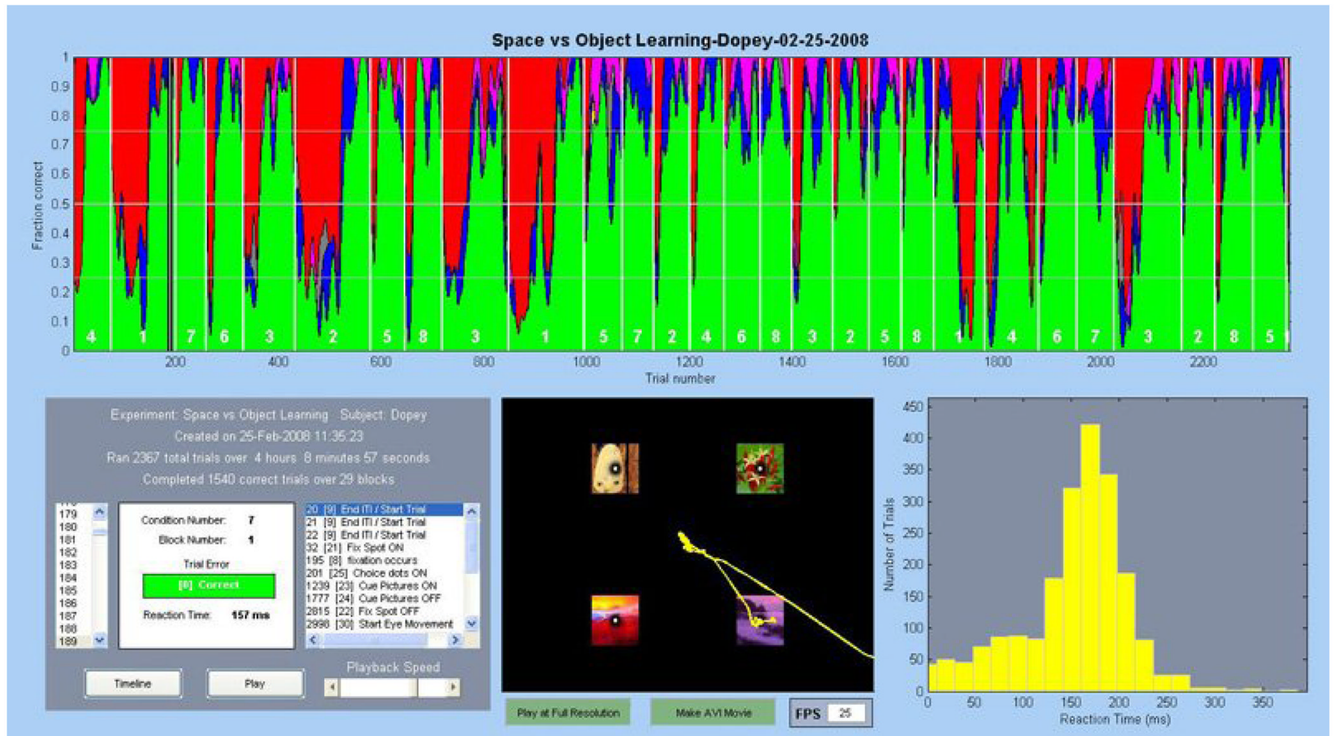
```
% Example Delayed-Match-to-Sample Task (dms.m)
% give names to the TaskObjects defined in the conditions table:
fixation_point = 1; sample = 2; target = 3; distractor = 4;
% define time intervals (in ms):
wait_for_fix = 1000; initial_fix = 1000; sample_time = 500; delay = 1500;
max_reaction_time = 500; saccade_time = 80; hold_target_time = 300;
% fixation window (in degrees of visual angle):
fix_radius = 1;

% TASK:
% initial fixation:
toggle(fixation_point);
ontarget = track('acquirefix', fixation_point, fix_radius, wait_for_fix);
if ~ontarget,
    trialerror(4); %no fixation
    return
end
ontarget = track('holdfix', fixation_point, fix_radius, initial_fix);
if ~ontarget,
    trialerror(3); %broke fixation
    return
end

% sample epoch
toggle(sample); %turn on sample
ontarget = track('holdfix', fixation_point, fix_radius, sample_time);
if ~ontarget,
    trialerror(3); %broke fixation
    return
end
toggle(sample); %turn off sample

% delay epoch
ontarget = track('holdfix', fixation_point, fix_radius, delay);
if ~ontarget,
    trialerror(3); %broke fixation
    return
end

% choice presentation and response
toggle([target distractor]); %simultaenously displays target and distractor
ontarget = track('holdfix', fixation_point, fix_radius, max_reaction_time);
if ontarget, %max_reaction_time has elapsed and is still on fix spot
    trialerror(1); %no response
    return
end
ontarget = track('acquirefix', [target distractor], fix_radius, saccade_time);
if ontarget ~= 1,
    trialerror(6); %incorrect response
    return
end
toggle(fixationpoint); %turn off fixation

% hold target then reward
ontarget = track('holdfix', target, fix_radius, hold_target_time);
if ~ontarget,
    trialerror(5); %broke fixation
    return
end
trialerror(0); %correct
goodmonkey(50, 3); %3 x 50ms of juice
toggle([target distractor]); %turn off remaining objects
```

Figure 4. Example Construction of a Simple Delayed-Match-to-Sample (DMS) Task

The over-all task design of a standard DMS task is shown in (a). The task consists of a fixation period, sample period, delay period, and then the presentation of choices. The subject's goal is to select that object among the choices that matches the sample cue, by making a saccade to that object. The subject must fixate on the central dot throughout the task until the choices are presented. (b) A conditions table describing this task. This table allows for either of two pairs of objects to be used on any given trial: pictures A & B, or pictures C & D. "Relative Frequency" determines how likely a particular condition is to be chosen, relative to the other conditions. "Conditions in Block" enumerates the blocks in which that particular condition can appear (for instance, running block #2 would play only conditions 5–8, and so would use only pictures C & D). "Timing File" refers to the Matlab script that organizes the contingencies that relate behavioral monitoring to stimulus presentation (as in (c), below). Here, all conditions use the same timing file. "Object" columns list the stimuli that can appear in each condition. These

can be visual objects, sounds, analog waveforms, or TTL pulses; any type of stimulus object can be triggered with the same *toggle* command. Note that, to simplify coding of the timing script, objects serving the same purpose are always in the same column (so, here, the sample object is always Object #2 and the target is always #3). (c) The Matlab script constituting the timing file used for this DMS task is shown. Functions that are provided by our software are highlighted in bold. Several other functions exist for time-stamping behaviorally-relevant events, repositioning-objects on-the-fly, turning the joystick-cursor on and off, defining interactive "hot keys" that initiate user-defined functions when the a key is pressed, etc.

NIH-PA Author Manuscript
NIH-PA Author Manuscript
NIH-PA Author Manuscript



NIH-PA Author Manuscript
NIH-PA Author Manuscript
NIH-PA Author Manuscript

NIH-PA Author Manuscript

NIH-PA Author Manuscript

NIH-PA Author Manuscript

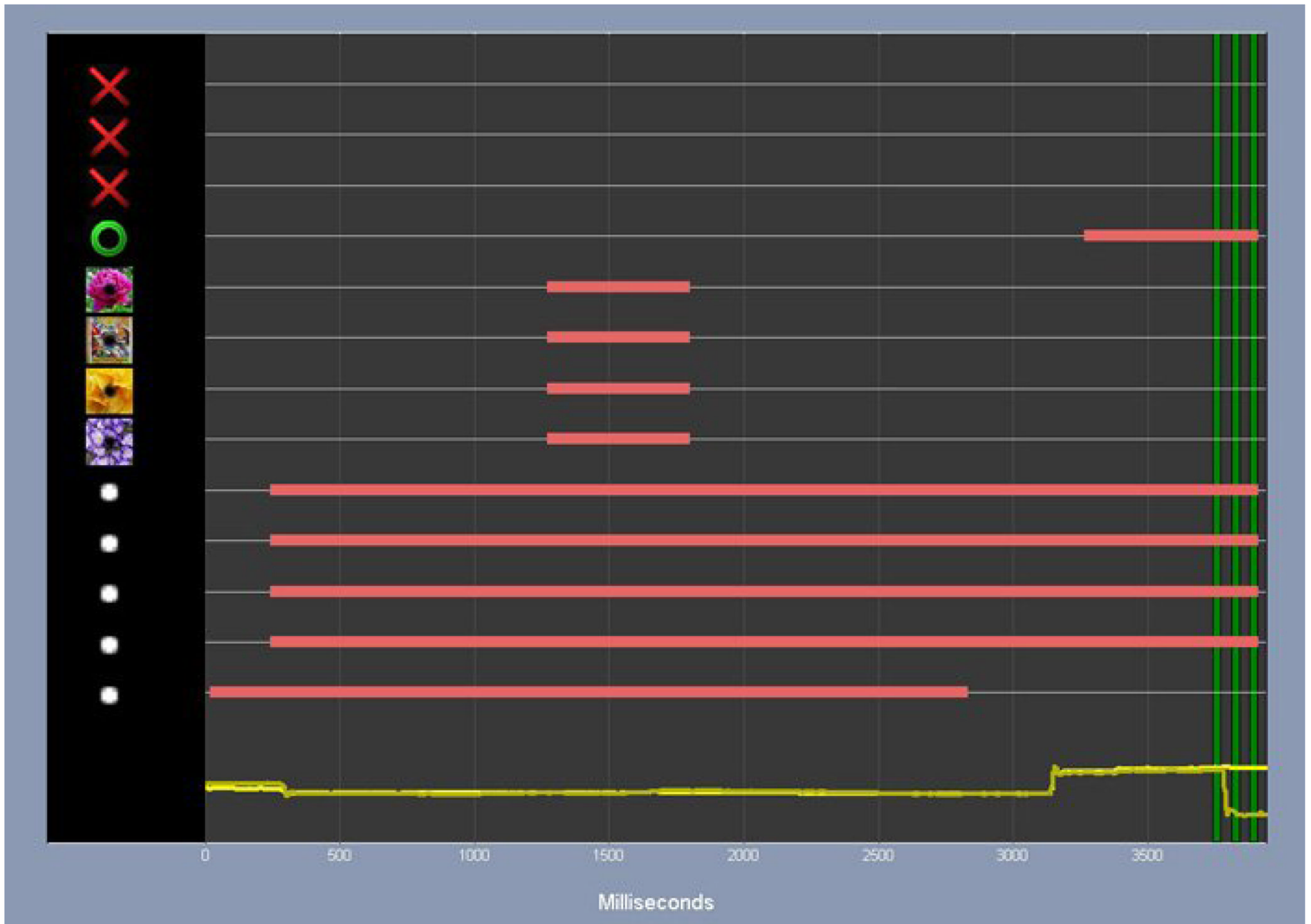


Figure 5. Behavioral Graphs

A basic overview of performance is shown in (a). Behavior over time is plotted at the top, reaction times are shown at the bottom-right, and basic file information and trial-selection is at the bottom-left. In the bottom-middle is an area which shows the objects used on the currently-selected trial and the eye- or joystick record. The trial can be played back in this window by pressing “Play,” and a movie can be created from any trial. (b) A time-line representation of the currently-selected trial. X- and Y-eye or joystick traces are shown at the bottom. The horizontal red bars indicate that the object at left was currently visible. The vertical green lines indicate reward delivery (here, three pulses of juice).

NIH-PA Author Manuscript NIH-PA Author Manuscript NIH-PA Author Manuscript

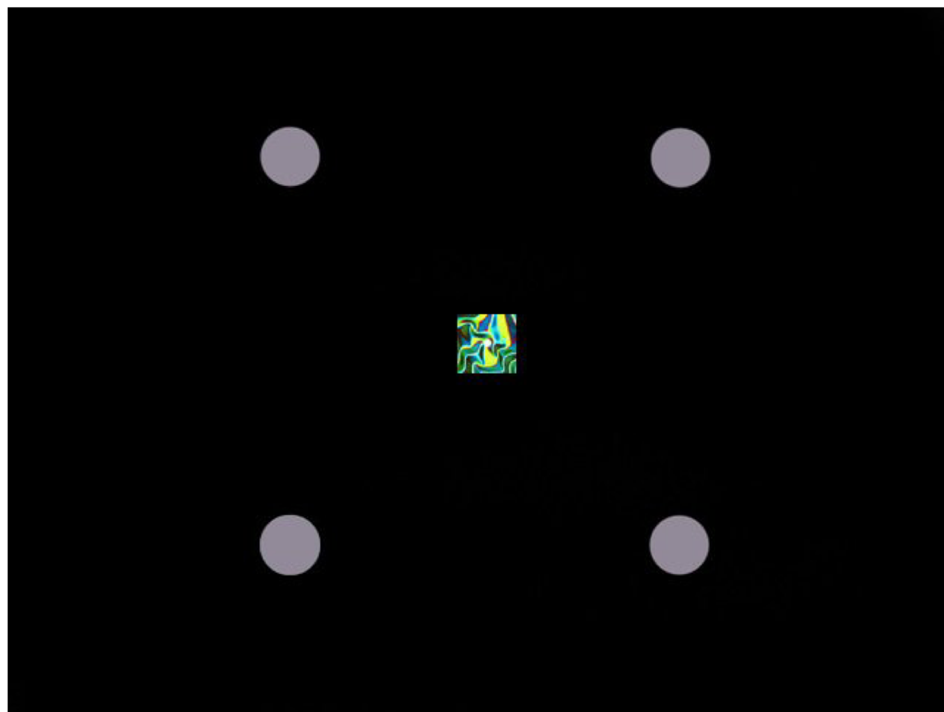


Figure 6. Experimenter's and Subject's Screens

The experimenter's display (a) contains a representation of the stimuli currently visible on the subject's display (b). In addition, a red ring marks the boundaries of the current fixation (or joystick target) window and a dot represents the current eye- or joystick-position (updates are generally set to occur every 50 or 100 ms, depending on user preferences). In the top-right, the experimenter's display relays information about the current trial, condition, block numbers and performance over-all, over the current block, and for the current condition. Reaction time histograms over-all and for the current condition are plotted at the lower-right.

NIH-PA Author Manuscript
NIH-PA Author Manuscript
NIH-PA Author Manuscript

NIH-PA Author Manuscript

NIH-PA Author Manuscript

NIH-PA Author Manuscript

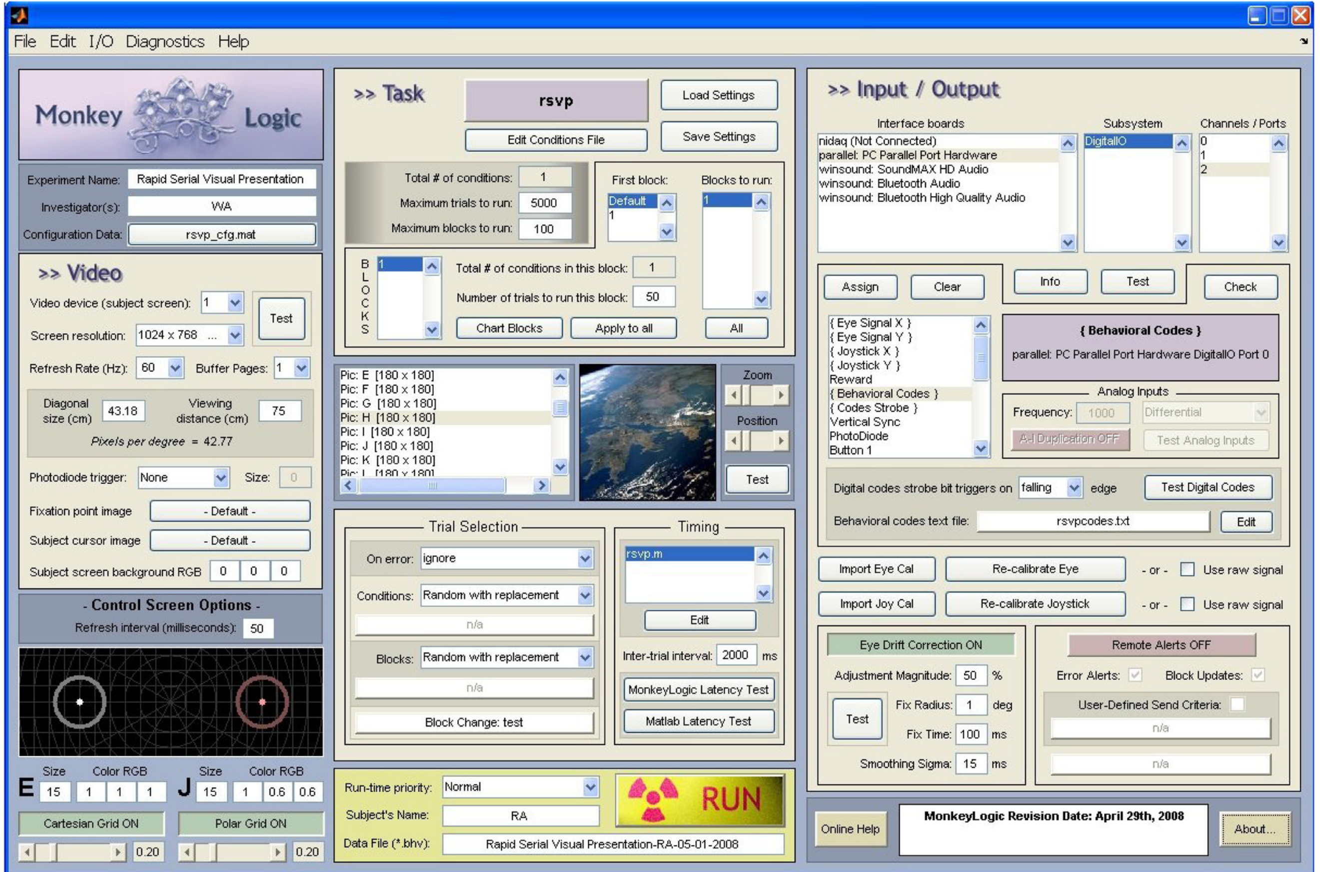


Figure 7. The Configuration Menu

This menu is where an experiment and all of its configuration parameters can be loaded, modified, and saved. Video settings are organized within the left panel. Input-output assignments and other settings are found within the right panel. Task-execution settings (e.g., condition- and block-selection criteria) are in the middle panel. This menu is the point from which an experiment is launched.

NIH-PA Author Manuscript NIH-PA Author Manuscript NIH-PA Author Manuscript

Table 1

Function Timing

The measured times based on 1601 trials of the sample task (see Methods) are presented. "Entry Time" refers to the amount of time required for initialization of each function, before the execution of the essential activity. The "Core Time" in each case reflects the amount of time required to execute that activity. Lastly, the "Exit Time" is the amount of time required to clean up and leave the function (i.e., return control to the user) after the core activity has completed.

For the "Trial" row, the entry time corresponds to the time required to initialize all the trial sub-functions (i.e., *toggle, track*, and *eventmarker*, as well as others), and to initiate analog data acquisition. The core time here would be wholly dependent the user's timing script specifying the timing and contingencies of the behavioral task itself, so this is not shown. The exit time reflects the time from the end of the user's script to the end of the trial, during which analog data and event markers are retrieved for storage in the local data file (event-markers were also sent to a separate neural data acquisition system in real-time).

In the case of the *toggle* sub-function, the entry time is the time required to parse the user's command options and blit the appropriate stimuli to the screen's back-buffer. Then, the core activity is flipping the screen to display or extinguish the requested stimuli. Note there will be a variable delay (excluded from the values shown here) between the completion of these entry tasks and the execution of the flip; the exact delay is inversely linearly dependent on the time remaining until the next flip at the time this function is called. The exit time is the time required to display the control screen symbols corresponding to the currently visible stimuli before returning control to the user.

In the case of the *track* sub-function, the entry time corresponds to the time required to parse the user's command options and calculate the target thresholds. The core activity consists of one cycle retrieving the most recent analog data samples, transforming these into calibrated coordinates, and comparing these coordinates against those of the possible targets. The exit time here corresponds to the time required to extinguish the target rings and return control to the user.

In the case of the *eventmarker* sub-function, the entry time is the time required to parse the user's command options and convert the decimal integers to binary form for digital output. The core time is the time required to write the digital bytes to the neural data acquisition system (two operations were required: first set the value bits, then trigger the strobe bit). The exit time is the time needed to reset the digital strobe bit, buffer the time-stamp for local storage, and to return control to the user.

The I.T.I. preparation time is the time needed to select the next trial according to the built-in block- and condition-selection options (a user-specified function could take longer or shorter), load the necessary stimuli from disk to video memory (here, six 100 × 100 pixel true-color images, each file 4 KB in size, as well as five program-generated fixation dots, 8 × 8 pixels in size, were used), and update the control-screen graphics to reflect the updated behavioral performance measures (e.g., percent correct over-all, perblock, and per-condition, and reaction times over-all and per-condition). The graphical updates constitute the bulk of this time (only ~2–4 milliseconds are required for trial selection and stimulus preparation under these conditions).

Function	Mean(ms)	Max(ms)		First Trial (ms)
Trial				
Entry Time	0.11	0.11		0.15
Exit Time	14.07	28.06		45.76
Toggle:				

Function	Mean(ms)	Max(ms)		First Trial (ms)
Entry Time	1.02	2.21		14.94
Core Time	0.18	0.29		0.91
Exit Time	25.85*	29.99*		26.60*
Track:				
Entry Time	1.15	1.29		1.16
Core Time**	0.98	2.27		2.04
Exit Time	1.09	1.24		1.09
EventMarker:				
Entry Time	0.24	0.35		4.30
Core Time	0.50	0.69		1.40
Exit Time	0.01	0.11		0.46
Inter-Trial-Interval:				
Preparation Time	99.51	115.51		234.43

* These exit times for the *toggle* function are modifiable by the user: one can elect to skip drawing the control-screen symbols corresponding to the currently active visual stimuli, in which case, the exit times averaged 0.91 ms (1.77 ms maximum). This option is useful when many stimuli must be presented in rapid-succession with precise timing, or when a ~25 ms delay before the next trial event is sub-optimal.

** The core times in the case of the *track* function shown here exclude the first cycle, during which the initial control-screen update is performed. This is the cycle in which rings around the specified targets are drawn to the experimenter's display. This one initial cycle took 22.0 milliseconds on average (26.7 milliseconds maximum).

NIH-PA Author Manuscript NIH-PA Author Manuscript NIH-PA Author Manuscript