



PLDAPS: a hardware architecture and software toolbox for neurophysiology requiring complex visual stimuli and online behavioral control

Kyler M. Eastman* and Alexander C. Huk

Neurobiology, Psychology, and Center for Perceptual Systems, The University of Texas at Austin, Austin, TX, USA

Edited by:

Markus Diesmann, Research Center Juelich, Germany

Reviewed by:

Graham J. Galloway, The University of Queensland, Australia

Lars Schwabe, University of Rostock, Germany

*Correspondence:

Kyler M. Eastman, Center for Perceptual Systems, The University of Texas at Austin, 1 University Station, A8000, SEA 4.328, Austin, TX 78712, USA.

e-mail: kyler.eastman@gmail.com

Neurophysiological studies in awake, behaving primates (both human and non-human) have focused with increasing scrutiny on the temporal relationship between neural signals and behaviors. Consequently, laboratories are often faced with the problem of developing experimental equipment that can support data recording with high temporal precision and also be flexible enough to accommodate a wide variety of experimental paradigms. To this end, we have developed a MATLAB toolbox that integrates several modern pieces of equipment, but still grants experimenters the flexibility of a high-level programming language. Our toolbox takes advantage of three popular and powerful technologies: the Plexon apparatus for neurophysiological recordings (Plexon, Inc., Dallas, TX, USA), a Datapixx peripheral (Vpixx Technologies, Saint-Bruno, QC, Canada) for control of analog, digital, and video input–output signals, and the Psychtoolbox MATLAB toolbox for stimulus generation (Brainard, 1997; Pelli, 1997; Kleiner et al., 2007). The PLDAPS (“Platypus”) system is designed to support the study of the visual systems of awake, behaving primates during multi-electrode neurophysiological recordings, but can be easily applied to other related domains. Despite its wide range of capabilities and support for cutting-edge video displays and neural recording systems, the PLDAPS system is simple enough for someone with basic MATLAB programming skills to design their own experiments.

Keywords: eye tracking, software, behavioral control, MATLAB, neurophysiology

INTRODUCTION

The rate of progress in cognitive and systems neuroscience is often constrained by the ability of a research group to design new experiments while taking advantage of advances in technology. In the last few years, several companies have made significant progress in providing user-friendly hardware and software for increasingly advanced forms of stimulus presentation, behavioral measurements, and neurophysiological recording. However, the task of integrating these experimental technologies has been left to the programming abilities of individual laboratories. This has led some laboratories to delay adopting these new technologies in order to avoid a set of daunting technical hurdles. Other laboratories have built custom, lab-internal systems that work for the range of experiments currently used by the lab. These systems become a burden when the primary developer leaves the lab, or the direction of research requires a significantly different experimental paradigm. Because of these issues, labs often are constrained to the reduced set of scientific questions that can be answered by small adaptations of their current experimental equipment. Primary investigators at the beginning of their careers are forced to “inherit” paradigms because of the burden of developing a new system from scratch.

As a solution to these challenges, our lab developed the Plexon–Datapixx–Psychtoolbox system (PLDAPS) (pronounced “platypus”) system. It is a MATLAB-based

toolbox for integrating several pieces of hardware and software (<http://hukdata.cps.utexas.edu/archive/PLDAPS.html>) already used in the behavioral and neurological sciences. PLDAPS was designed to provide developers with a basic knowledge of MATLAB the tools necessary for designing their own electrophysiological experiments. More specifically, we wished to develop a system that would be used by researchers to extend the well-established Psychophysics Toolbox (Brainard, 1997; Pelli, 1997; Kleiner et al., 2007) into electrophysiological research. The technical issues of coordinating neurological and behavioral events, file management, and stimulus generation are taken care of in a transparent way, allowing the developer to focus on their experimental design. In this report, we show how PLDAPS integrates information from the three hardware components, demonstrate some of the features that enhance programming flexibility, and show that it can be used as a finished system in the context of single and multi-unit electrophysiology in awake, behaving primates.

The following sections give an overview of PLDAPS, review the three motivating but conflicting goals, and explain how specific features are designed to accommodate these goals. Then, specifics of the system are described in terms of hardware and software, results of some timing performance tests are reported, and solutions to some technical challenges are summarized. Finally, PLDAPS is compared to other existing systems, and possible future applications are explored.

MOTIVATING GOALS OF PLDAPS

Plexon–Datapixx–Psychtoolbox was developed in response to the problem of simultaneously satisfying three separate and conflicting goals: (1) usability, (2) programming flexibility, and (3) temporal precision. Each of these goals may have been individually satisfied by existing systems. However, in doing so, their designs have led to compromises in the other two. PLDAPS's novel contribution is that it satisfies all of them with few sacrifices. Moreover, these goals are in addition to the basic function of PLDAPS, which is to coordinate stimulus generation, behavioral monitoring, and electrophysiological recording. The PLDAPS system uses a single Macintosh Pro running MATLAB, enabling behavioral monitoring and stimulus generation to happen almost concurrently, on a single computer, in a single set of MATLAB functions (as opposed to being split, for example, between an experimental-control computer and a video display “slave” computer). In addition, a major challenge of any such system is the ability to acquire a wide set of signals and events, from continuous analog signals (e.g., eye position, motion tracking, electromyography) to digital records of experimental “events” (e.g., the onset of a visual stimulus, the detection of a behavioral responses). These need to be temporally co-registered with the video display and the neurophysiological signals of primary interest. In the PLDAPS system, this analog and digital data input (and output) is handled with high temporal precision by a Datapixx input/output device and controlled (i.e., polled input or controlled output) by low-level MATLAB functions supported by the PsychToolbox and further facilitated by our PLDAPS software toolbox (see **Figure 1**).

PROGRAMMING FLEXIBILITY

The issue of programming flexibility, i.e., the speed and extent to which current experimental setups can be adapted to new paradigms, is a major problem that can inhibit the rate of scientific production in both individual labs and in the field of neuropsychological research in general. PLDAPS was intended to address this issue by taking advantage of the popularity and functionality of MATLAB, a high-level programming language that has replaced lower level languages like C/C++ or Java as a common language for experimental design and data analysis. In fact, all three components of PLDAPS include MATLAB-based functionality; both Plexon and VPixx technologies provide Toolbox interfaces with their hardware, and Psychtoolbox has been refined over many years to provide MATLAB users the ability to program increasingly sophisticated psychological experiments. Thus, the main design goal of PLDAPS was to simply preserve the functionality that each of these components already provide. By recognizing that more and more graduate students are required to already know MATLAB to perform data analysis and behavioral experiments (with Psychtoolbox), PLDAPS leverages that knowledge into the ability to design and implement electrophysiological experiments.

Prior experimental systems have often emphasized usability (i.e., a robust GUI) at the expense of programming flexibility, and/or have assumed that such flexibility would come at the cost of a loss of temporal precision. By relying on modern computing hardware and a high-level programming language, PLDAPS can simultaneously support usability, flexibility, and temporal precision. This is because solutions for individual experimental needs

can be easily and quickly prototyped, iterated, and implemented – all without needing to delve into low-level aspects of hardware I/O.

For example, consider one experimental issue, in which the size of an eyetracking target window often needs to be adjusted in the middle of an experiment. The experimenter can create a user interface where that window size parameter can be adjusted by editing a value in a MATLAB m-file, or even by interacting with a GUI slider button. Now consider a large number of such experiment-specific needs, all of which together form an interface that consists of a variety of experiment-specific controls, along with some controls that overlap with the needs of other related experiments. When a new experimental procedure is required, less flexible systems require one of two possible directions: either work with the existing interface or create a new one. The former solution often exposes a lack of flexibility and thus leads to *ad hoc* changes, leading to an amalgam of idiosyncratic or unused code, and interface detritus; the latter solution requires significant coding and consideration of low-level hardware–software interface. These issues become more severe when the primary developers (most often graduate students) change labs or universities. For this reason, PLDAPS was designed to be built with a minimal amount of paradigm-specific architecture, but its MATLAB backbone allows for easy sharing of modular pieces of code.

USABILITY

While the primary design of PLDAPS provides the developer with the ability to program a wide range of experiments, that capability is irrelevant if the experimental programs are not usable enough to carry them out. Electrophysiology sessions on awake-behaving monkeys are unusual in that the experimental paradigms require a high amount of cognitive load on the experimenter. The experimenter has to monitor a range of concerns, such as the welfare of the monkey, the functionality of the behavioral monitoring (such as the eyetracking signal), and the status of the electrode signal, which may be compromised due to a wide range of electrical and mechanical issues. On top of this monitoring, the success of the experimental session is dependent upon a series of subjective perceptual decisions: to judge that a cell is task-related, has a distinctive receptive field, is capable of being isolated, or that better cells may exist along the electrode path. These judgments are multiplied when the paradigm attempts recording from multiple neurons.

The cognitive burden on the experimenter can be exacerbated by the opaque nature of the system interface. Many of the current systems require opening and closing multiple windows to adjust a single parameter, many of which need to be adjusted dozens of times during an experiment. For this reason, PLDAPS must be able to have these adjustments made quickly and in a straightforward manner. Furthermore, perceptual judgments that affect the outcome of the experiment, such as the location and shape of a receptive field, are usually done *ad hoc*, partly due to the constraints of the software. Now that more programming flexibility is provided from PLDAPS, graphs, and scopes of behavioral and neural processes, or other decision aids, are no longer on a wishlist but are instead capable of being quickly developed on a paradigm-by-paradigm basis.

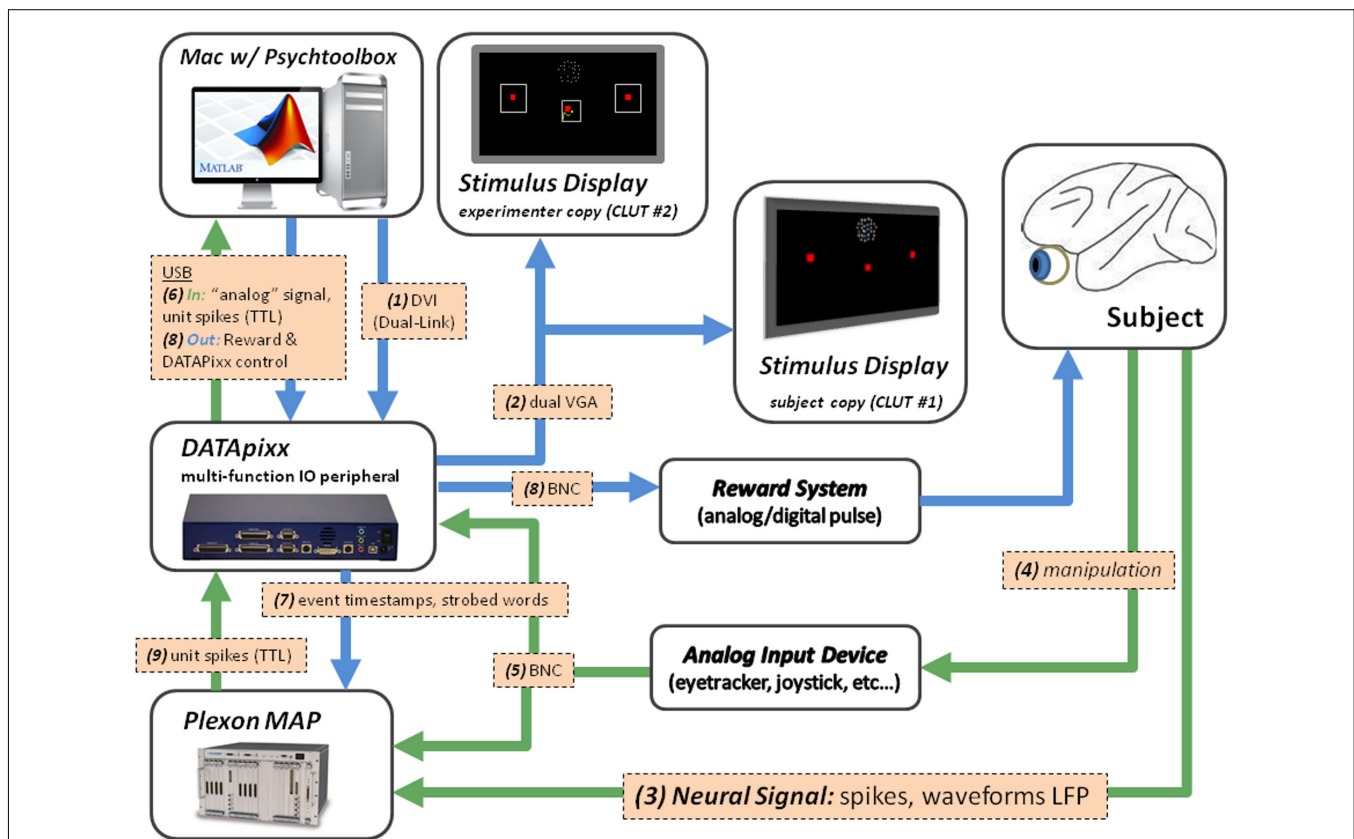


FIGURE 1 | Roadmap: an overview of the information flow within PLDAPS. In addition to the subject (which presumably has a brain), there are five components: a Plexon MAP device, the Datapixx USB peripheral, a Mac computer running Psychtoolbox, a reward dispensing system, and an input device (such as a keyboard, joystick, or eye tracker). (1) The Mac, running an experimental program in Psychtoolbox, sends digital video information to the video buffer on the Datapixx peripheral. (2) The Datapixx box splits the video information to two displays, made slightly different by two different color look-up tables (CLUT). One screen is for the subject, while the other has additional window and input markings (such as eye position) for the experimenter to monitor the subject's activity in real-time. (3) The subject, responding to the screen, generates neural signals that are recorded by the

Plexon MAP. (4) The subject also manipulates the input device. (5) Redundant analog signals are sent both to the Plexon device and the Datapixx box. (6) The Datapixx box sends the subject's input back to the Mac, where the experimental program updates events in the trial (such as fixating on a target). (7) Timestamp markers of events in the trial are sent via TTL pulses to the Plexon MAP via an 8-bit digital cable from the Datapixx. At the end of the trial, more complex trial information is transmitted via 8-bit strobed words. (8) An analog or digital pulse is sent to the system that dispenses a reward. (9) An option is to send TTL pulses of neural spikes to the Datapixx box. This is used to closed-loop experiments where experimental parameters for the next trial are determined from neural activity in the previous trials.

TEMPORAL PRECISION

An increasing amount of temporal precision is required by the line of current experiments. Plexon, Datapixx, and Psychtoolbox have devoted a considerable amount of time to maintaining precision. Therefore, the design goal of PLDAPS is to maintain this precision during the coordination of the hardware systems. While these systems are faithful to their internal clocks, as we show later with performance tests, the internal clocks have enough of a difference to require care when the timestamps of those clocks are coordinated.

PLDAPS FEATURES

When designing PLDAPS, we initially identified key design features that enabled PLDAPS to satisfy the goals described above. None of these features map on to a single design goal. Instead, each of them represents ways to optimize the overall utility

of PLDAPS by providing the best compromise across multiple goals.

TOOLBOX FORMAT

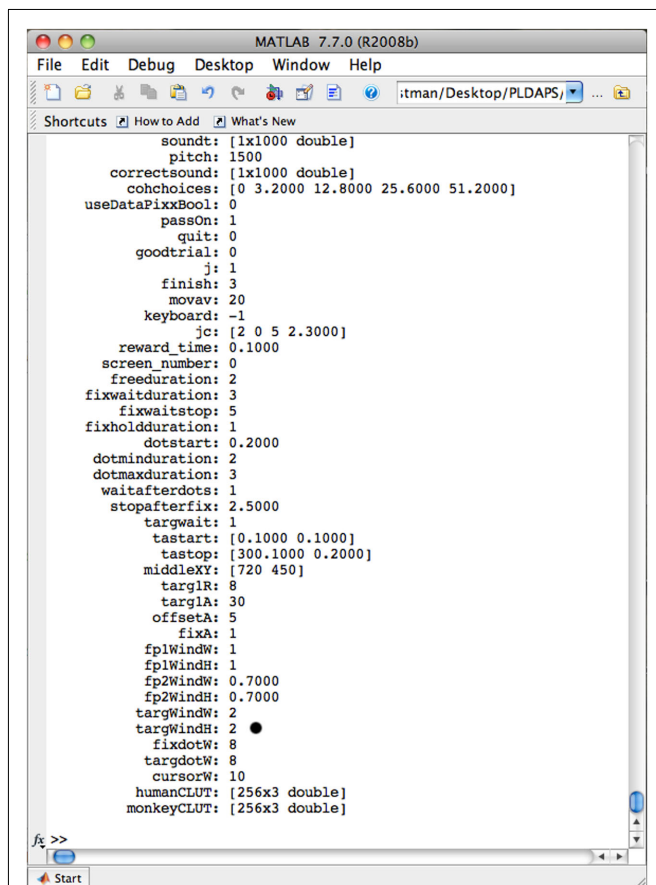
While we intended to build a functional electrophysiology system, we also intended PLDAPS to be a library of functions for others to combine in their own programming efforts. The most straightforward path to this end was to develop PLDAPS in the MATLAB toolbox format that MATLAB users are already familiar with. The main component of the PLDAPS toolbox is a MATLAB script, such as **letsgorun**, with three basic sections: code that manages experiment files, a function that gets the Datapixx box ready, and a WHILE loop that iterates a trial function, which iterates a single trial. Two different functions are included in the toolbox, **runMapTrial** (runs a mapping trial, e.g., for localizing a visual receptive field),

and `runDecisionTrial` (a two-alternative forced choice visual discrimination task).

COMMAND-LINE INTERFACE

The PLDAPS system consists of a layout for connecting several pieces of hardware, and a set of sample scripts and functions that demonstrate how they can be functionally integrated. This leaves the specific computer programs for running experiments to the individual. In contrast to many current options, we chose not to develop a graphical interface, which often limits the capacity of the user to make changes, and also removes them from the actual underlying code. Thus, the PLDAPS system returns the actual programming of the visual stimuli and experimental logic to the experimenter, while still supporting the integration of multiple hardware systems.

In PLDAPS, a real-time display of all the experimental parameters is generated by outputting the structure defined in the condition file after every trial (see **Figure 2**). Although this is (intentionally) crude, it avoids the potential coding burden or loss of flexibility associated with a graphical interface for a specific application (users can easily build GUIs on top of this structure).



```

MATLAB 7.7.0 (R2008b)
File Edit Debug Desktop Window Help
Shortcuts How to Add What's New
soundt: [1x1000 double]
pitch: 1500
correctsound: [1x1000 double]
cohchoices: [0 3.2000 12.8000 25.6000 51.2000]
useDataPixxBooL: 0
passOn: 1
quit: 0
goodtrial: 0
j: 1
finish: 3
movav: 20
keyboard: -1
jc: [2 0 5 2.3000]
reward_time: 0.1000
screen_number: 0
freeduration: 2
fixwaitduration: 3
fixwaitstop: 5
fixholdduration: 1
dotstart: 0.2000
dotminduration: 2
dotmaxduration: 3
waitafterdots: 1
stopafterfix: 2.5000
targwait: 1
tastart: [0.1000 0.1000]
tastop: [300.1000 0.2000]
middleXY: [720 450]
targlR: 8
targlA: 30
offsetA: 5
fixA: 1
fp1WindW: 1
fp1WindB: 1
fp2WindW: 0.7000
fp2WindB: 0.7000
targWindW: 2
targWindB: 2
fixdotW: 8
targdotW: 8
cursorW: 10
humanCLUT: [256x3 double]
monkeyCLUT: [256x3 double]
fx >>
Start

```

FIGURE 2 | Command Window User Interface. The condition structure is displayed in the command window at the end of each trial, clearly displaying all necessary information. On-the-fly changes to individual fields are made by pressing “p” (for pause), which keyboard access. Typing return will continue the experiment.

Any parameters that are defined during experiment prototyping will be displayed during the experiment. If the experimenter wants to change them in the middle of the experiment, he or she starts by pressing “p,” which pauses the experiment. It also gives a keyboard prompt. The experimenter can then change any of the parameters in the condition structure. For example, to change the size of a window width, one would type in `c1.targWindW = 30` and then type “return.” This process can be accelerated by scrolling through commands with the UP arrow key. If a more substantial adjustment needs to be made, the experimenter types “q” for quit.

Graphical displays of behavioral or neurological data can be developed as needed. For example, if one would want to develop a display of the percent correct out of the last 10 trials, one would only need to insert the code `plot [smooth(PDS.correct,10)]` at the bottom of the WHILE loop. However, making such displays may introduce some timing variability, and are best kept in between trials. In **Figure 3**, we show an example of an interactive display. This “RF scope” was designed to help with the perceptual decision-making while experimenters are searching for a task-related cell with a distinguishable receptive field. A measure of selectivity is computed for each trial. A Gaussian process regression of all selectivity measurements is plotted as a heat map. The slider on the right allows the user to select a threshold on the map. The center-of-mass is calculated on the surface above the threshold, giving an approximation of the center of the receptive field. To insert this code, two functions were developed: `initRFmap`, and `updateRFmap`. Such a scope was developed for a specific experiment in less than thirty minutes.

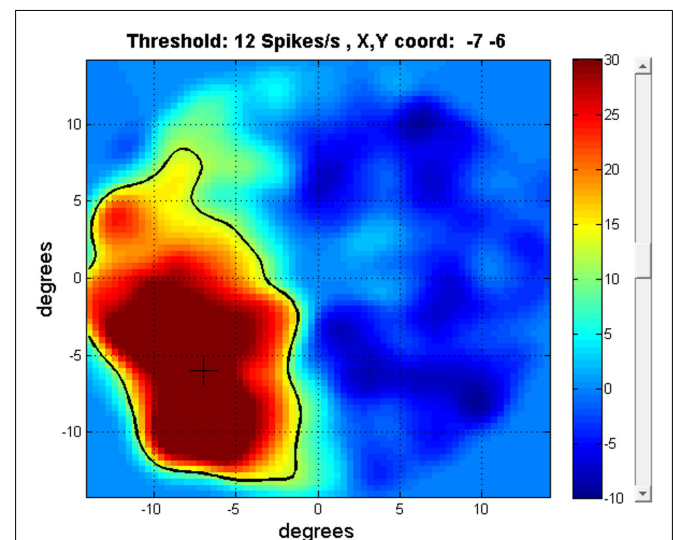


FIGURE 3 | Receptive Field Map. This graph shows the spatial selectivity (pre-saccade firing rate-base firing rate) of a neuron during each trial as a function of the target location. As the plot updates after each trial, it allows the experimenter another method of evaluating the size and shape of the neuron’s response field during the experiment. A threshold (black contour line) for calculating a center-of-mass for RF is movable with the slider button on the right hand side. Such interfaces can be developed and customized quickly according to the experimenter’s tastes.

MULTIPLE-DISPLAY, MULTI-CLUT SYSTEM

It is often helpful for the experimenter to be able to view a copy of what the experimental subject is viewing. However, it is often critical for them to also have a “behind-the-curtain” view of this; for example, to know where the subject’s eye position is relative to desired spatial “windows,” where the location of the correct response target is on each trial, etc. Usually this requires two screens: one is a direct, mirrored copy of the actual experimental video display, and the other is an iconic representation of the underlying “semantics” of the task relative to the stimuli. The PLDAPS system solves this in an elegant manner using two different color look-up tables (CLUTs). Specifically, the Datapixx box splits the video signal, and one copy is shown to the experimental subject, and the other is viewed on a separate monitor by the experimenter. However, the two displays have slightly different CLUTs, allowing the experimenter’s display (but not the subject’s) to also animate the online eye position, target windows, and other semantics. This is accomplished by setting the corresponding CLUT entries for the subject to be the same as the background color, and also by drawing the experimenter-specific elements first, and letting the “real” stimuli be drawn over them. This ensures that the actual stimulus never shows the experimenter-specific elements, nor is ever occluded by the experimenter elements. Furthermore, this trick is computationally efficient: the Macintosh simply generates one copy of the stimulus (including experimenter-view elements), and the two distinct views arise solely from the difference in the CLUTs. Thus there is no need to create two parallel views, reducing computational and animation resources. In our experience, a single modern Macintosh Pro can both generate very advanced graphical stimuli and control the experimental state logic at conventional video frame rates.

MATERIALS AND METHODS

ARCHITECTURE AND COMPONENTS

The essential components of the PLDAPS system consists of a Macintosh Pro computer for stimulus generation (“Mac Pro” hereafter), A VPixx technologies Datapixx USB I/O peripheral (“Datapixx box”), a Plexon electrophysiological recording system consisting of a multi-channel acquisition processor (MAP) and a Dell data management computer (“Plexon MAP”), and input devices, such as eye trackers, joysticks, etc. The general architecture of the system is outlined in **Figure 1**.

Computer specifications

The stimulus generating computer was a Mac Pro with an Intel Quad-Core Xeon processor running at 2.66 GHz and containing 16 GB of DDR3 RAM (Apple Inc., Cupertino, CA, USA). The operating system was Mac OSX Version 10.5.8. The graphics card was an NVIDIA Quadro FX 4800 with 1.5 GB of GDDR3 SDRAM. MATLAB (version 2010a, MathWorks, Inc., Natick, MA, USA) was used to write the PLDAPS toolbox, and to run timing tests. Stimulus generation was carried out through Psychtoolbox Version 3.0 (Brainard, 1997; Pelli, 1997; Kleiner et al., 2007).

The Datapixx box

A central component of our system was the Datapixx multi-function data and video processing USB peripheral (VPixx Technologies). The device contains a dual-display video processor and

both digital (16-bit) and analog (16 ADC and 4 DAC) I/O channels. Because the video controller and I/O control is on the same circuit board, the Datapixx box can synchronize inputs and outputs to the video refresh. In addition, a Datapixx software toolbox comes bundled with Psychtoolbox 3.0 and is also available on the VPixx website. This toolbox allows complete control over all functions of the Datapixx box within Matlab. The multi-CLUT system required a special firmware update from VPixx.

The Plexon MAP system

The Plexon MAP was used for all spike monitoring functions which interfaced with A WS-T3400 Dell Precision T3400 Workstation. This workstation had a 3.33-GHz Core 2 Duo processor. A PBX3/16sp-r-G1000 Preamplifier box was used for neural signal pre-amplification. A HST/32V-G20 headstage unit was used for simulated neural recordings. In addition, a National Instruments A/D subsystem plus accessories (C-HUB, 2 m CBL/C-HUB cable, and PCI-6071e A/D board) for 64-channel analog recording (1.25 MS/s, 12-bit resolution) were used. The SortClient software application was used during prototyping. To simulate a typical spike signal, a WAV file was played from the Plexon computer through the headstage unit via the headphone jack. To use our communication protocol between the Datapixx and Plexon, we chose to configure our digital input to allow 8 bits of time stamp information and 8 bits of strobed words (mode 2) (See Web References for details). To enable closed-loop experiments, a custom DB-15 to BNC cable was fabricated to enable real-time neural (spike) event marking (Plexon offers other hardware solutions for this application).

Video display specifications

There are no firm constraints on video displays. In our development and testing, three (rather different) monitors were used: a 21” Dell p1130 CRT display with a resolution of 1280 × 1024 pixel resolution at 75 Hz, a Dell 2208 WFPt Flat Panel Display at 1680 × 1950 pixel resolution at 60 Hz, and a 55” LG LH90 1080p 1920 × 1080 pixel resolution display running at 60 Hz.

Other input devices

During development and testing, we used an analog joystick as a proxy for any sort of device (e.g., an eye tracker) that would provide analog outputs to the system. The voltage ranges were from (0–2/5 V *x*-axis, 2.5–5 V *y*-axis). The ease of incorporating other input devices is discussed in the discussion section.

Experimental setup

Almost any experimental paradigm has a list of parameters that are repeatedly used for experimental sessions, but are often copied and adjusted over time to create multiple conditions, additional experiments, etc. Our experimental design is no different, with all relevant parameters listed in a MATLAB condition structure, C. The elements of this structure are defined in a single file that, in total, represents the conditions of a particular experiment. Since this file is a MATLAB “m-file,” it can be modified in any text editor.

Likewise, almost any experiment requires a file of recorded data. Our data file, with the extension “.PDS,” lists all of the data as a single MATLAB structure (in a conventional MATLAB “.mat” file

format). This allows different kinds of data (with different sizes) to be labeled and organized by trial. For example, a PDS structure for 100 trials of a two-alternative forced choice task can contain both PDS.choices [a 100×1 boolean array representing the left (0) or right (1) choice], as well as PDS.Spikes (a $100 \times$ variable length cell array that contains timestamps for all spikes associated with each trial). Furthermore, the structure format allows simple extension to a multi-session analysis; a multi-PDS array can easily be constructed of PDS structures by looping over individual PDS sessions.

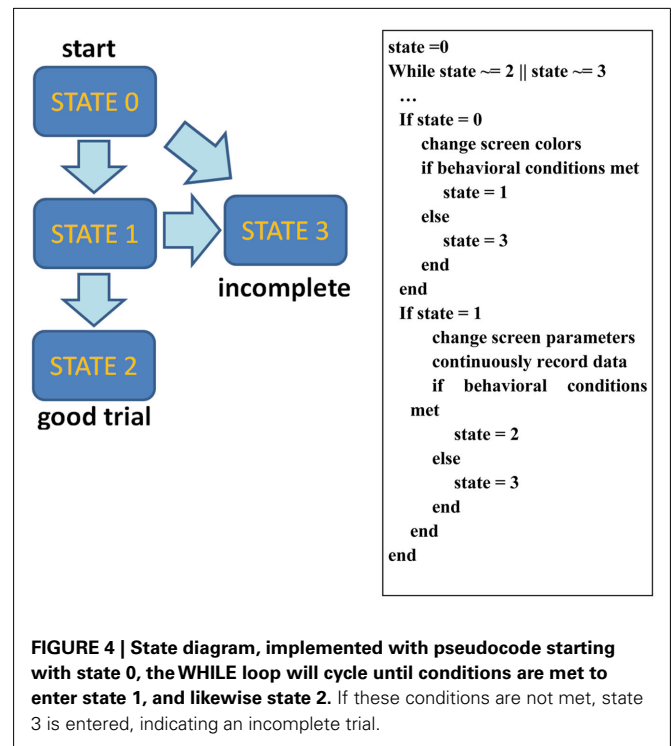
It is common for neurophysiological experiments to be stopped and re-started in the middle of a session, often during challenging points of the experiment that require on-the-fly adjustments by the experimenter. It is therefore critical that such a system be robust to such stop-start use, maintaining the integrity of data files. The PLDAPS toolbox saves MAT files that contain both the C (Condition) and PDS (data) structures. If the experiment is stopped and then re-started, a dialog box prompts the experimenter for either a new experiment name or an existing one. If an existing PDS file is used, the experiment will continue from where it left off.

Contingencies have been developed for an event that abruptly stops PLDAPS from functioning, such as a power outage. To preserve the existing data, small temporary files are saved to a location defined by the user (potentially off-site). Each of these files is essentially a PDS structure of each individual trial. Temporary files can be gathered using the **gather** script.

Experimental function architecture

We chose an architecture with many of the conventions of finite-state machines (see Wagner, 2006) with starting states, entry, exit, input, and transition actions. In this case, the states are time periods where the stimulus program enters and exits depending on particular actions, such as an eye position entering into a window. These conventions are still implemented with MATLAB procedural code, with a WHILE loop, and a state variable that gets changed within a series of modular conditions IF statements. As the WHILE loop iterates, it checks for certain behavioral conditions (such as an eye position entering a window), records relevant data, and continually updates the screen (such as the next frame of a movie). Starting in state 0, it waits until the conditions are met, and, if they are met, will simply update the state parameter to state 1. A pseudocode example is shown in Figure 4. This is done with an IF statement containing IF state = N (or in MATLAB, state = N). Though the states could have string names, we used numbers to indicate a sequential order. Within each IF statement, there just needs to be an assignment to the next state (state = N + 1) when conditions are met. With a series of independent IF statements, only the code within the statement will be implemented in each cycle of the WHILE loop, allowing a modular design with each IF statement acting as an independent state.

On a modern Mac Pro, the WHILE loop cycles at a rate >10 kHz, a rate significantly faster than both the screen refresh and what is necessary to record our behavioral data. For this reason, we include two different conditional statements that regulate the speed of the recording and screen updating. To minimize computational cycles and to simplify coding, all semantic windows and markers were drawn on the screen for every screen flip. To make



the window invisible for a particular state, a marker was drawn as black, thereby matching the background.

The types of conditions that allow state transitions are only limited by the experimenter's imagination. However, the ones that are currently used are three types: a button press, an eyetrack or joystick window, or a time elapse. All three of these contingencies take advantage of variables that are updated at the end of each while loop, the time at the start of the trial, the current eyetrack/joystick position, and a current keypress (used by the PsychToolbox function **KbCheck**). Most timing contingencies, however, are usually defined in reference to the start of each state, which requires that to be checked as a variable. For example, one may want to define a time limit for a subject to fixate on a cue, which has only appears at the start of state 1. In this case, one would use a $current_time > enter_state1_time + state_1_start$.

Of course, there is no reason that each state has to occur entirely within a sequential order. It is possible for a state to be looking for multiple sets of behavioral conditions and to assign the next state. For example, if you have a state that is waiting for a decision indicated by a button press, the next state can be determined by whether the subject made a correct or incorrect response. These next two states, designating correct and incorrect answers, could have reward or punishment functions.

For the trial to end, one must specify one or more end states. These states are different in that they often do not require any separate lines of code. These states are simply listed in the conditions of the while loop, such as WHILE state $\sim=$ 3. In our case, we use two states, 3 and 4, to indicate whether a trial was incomplete or complete, respectively. Assignments to state 3 can be made throughout the trial code when a subject fails to complete the task needed to go to the next state.

Some contingencies are not defined as the requirements to enter the next state, but instead defined as the requirement to exist in the state itself. For example, in an eyetracking experiment, one must often maintain fixation, defined by keeping the eyetracking position within a defined window around a target. This leads to a particular IF, ELSE statement that would include an assignment to state 3. Some contingencies are used in conjunction. For example, to maintain fixation for state 1, one must keep within a specified window for a set of time.

Communication with Plexon

PLDAPS scripts communicate with Plexon via the Datapixx's digital output cable. This cable has 16 digital channels, eight of which are devoted to timestamp information; the other eight are devoted to strobed words (this is configured in the Plexon system). Timestamps are sent to the Plexon MAP server to make redundant event information that has a high degree of temporal accuracy (the precision of which is defined by each system). Strobed words are transmitted at the end of the trial to encode more complex information about each trial. The precision of each strobed word is limited by the 8-bits of information. However, multiple bits can be used for each data-point, at the discretion of the experimenter. Timestamps and strobed words are implemented with the `FlipDataPixxBit` and functions, respectively.

`FlipDataPixxBit` is a simple function inserted into the main experiment script that marks where an event occurs, such as the eye position moving within a target window. The only thing it does is send a pulse along a specified channel from the Datapixx Box to the Plexon, which marks the rising edge of this pulse.

For decision trials the channels have been marked according to the scheme described in **Table 1**.

`DataPixxStrobe.m` gives a number between 1 and 256 by sending it along in binary across the eight channels. This function is being used to send trial information (currently: a unique identifier, trial number, good trial, coherence, correct, in RF, answer). This is done at the end of the trial, which also conveniently marks the trial's end. Since each strobe can only output an integer 1–256, one has to be deliberate when defining real numbers with more precision. Booleans or indices of a pre-defined array are much easier. The current unique identifier is the output from MATLAB's clock function, which is convenient since month, day, hour, minute, second are all less than 256. Both the Datapixx and Plexon strobe much more data, up to 1000 words/s.

The communication protocol is intentionally designed to be redundant, duplicating timestamp information in the Plexon PLX file and the PDS file on the Mac stimulus machine. This is especially the case if spike information is being conveyed via event TTL pulses from the Plexon, which are read as spikes by the Datapixx and added to the PDS structure at the end of every trial. However, there may be reasons where this is not possible. Therefore, to combine the spike timestamps to the PDS data afterward, a script (`combine`) was used that integrates the Plexon PLX file and the PDS file, using a unique identifier (a vector from "clock") to align the trials. PDS files include all trials from the start of the experiment and PLX files will likely include a subset of these trials (because it is common to have the subject perform the task while

isolating a neuron). The `combine` script can integrate in the PLX and PDS files either by including or omitting the extra trials.

Hardware interfacing

A wide variety of hardware interface options are afforded by the Datapixx box since it includes both digital and analog inputs that can deal with a range of voltages and protocols. For example, both an analog eye tracker and a joystick can be used by changing the voltage levels on the A to D board. Also, a reward system for our subject (a small squirt of water) was implemented with the Datapixx box sending a TTL pulse to a valve solenoid.

RESULTS

PERFORMANCE TESTS

All three components (Plexon Datapixx, Psychtoolbox) have been designed to maintain a high amount of temporal resolution and accuracy. The Plexon system has been developed with specific hardware to allow multiple neural signals at microsecond accuracy (Plexon, Dallas, TX, USA). The temporal performance of Psychtoolbox been well-documented (Brainard, 1997; Pelli, 1997; Kleiner et al., 2007), and many techniques have been developed to maximize its performance (see Web References). In this section, we will describe the temporal performance of our system and the techniques we used to maximize PLDAPS's capabilities.

Co-registration of events on the Mac and in Plexon

To test the co-registration of a PLX and PDS file, a sample dataset was generated of the decision task previously described and implemented in `RunDecisionTrial`. The parameter `c.PassOn`, was switched to 1, which allows the experiment to "pass on" through each state, simulating trials without having to provide input. Three hundred fourteen trials were recorded with the Plexon MAP server recording a single well-isolated neuron (as approximated by a sample WAV file). Both a PLX and PDS structure was generated (automatically by the `combine` script). In the `combine` script, the spike timestamps are added to the PDS structure while the event information is preserved. The differences in the event timestamps had a mean of 0.13 ms, a SD of 0.4 ms, and an absolute max of 1.1 ms.

Co-registration of timing of events and actual display on the monitor

The speed requirements of a visual stimulus generation program are normally capped by the fact that most displays refresh at a rate of 60–120 Hz. Psychtoolbox programming works by calling functions that change draw object images and shapes in a buffer, which changes the pixel information in video memory. To update the screen, the "flip" function displays the video memory on the screen on the next available vertical retrace. So, generating the stimulus can often be performed on-the-fly, as long as the manipulations to the video memory take place before the next retrace. If this is not the case, pre-computing this information and storing it in RAM or in video memory is usually an acceptable alternative.

Our prototypical experiment was a two-alternative forced choice decision task with a random dot kinematogram (Huk and Shadlen, 2005). Our visual stimulus was relatively simple, consisting entirely of small dots for the subject, with the addition of a three rectangles to display windows for the monitor. To slightly

increase the efficiency, all dots were generated using one call of the Drawdots function. To avoid the need to examine the timing performance of each experimental state individually, all items were drawn continuously for all states, only changing colors (this also greatly simplified the code).

As noted in the design section, a trial function cycled continuously through a main WHILE loop. With our hardware set up (see Materials and Methods), the cycle rate of this loop was 10 kHz, which was far faster than both the refresh rate of the of our screens (60–75 Hz) and the sample rate requirements of our input (about 1 kHz). However, a screen refresh takes about 3-ms, depending on the computer and monitor. Furthermore, the PTB3 screen FLIP function (dependent on settings), once called, will wait until the next available screen refresh, which prevents the computer for further monitoring. To free the computer to sample at a higher rate, a conditional if statement (time of last screen refresh > specified interval) was inserted in the WHILE loop. To allow an adjustable sample rate, a similar IF statement was included. Both of these statements created a small amount of unavoidable variability in the sample rate (see Figure 5). These two rates were adjusted to avoid missed flips (and constitute the only "magic numbers" in the PLDAPS toolbox).

A systems timing test was conducted on three monitors (Figure 6) was conducted for two reasons: (a) length and variability of the screen refreshes and the sample rate variability for each monitor, (b) the time lag between the results of the photodiode and the screen. To do this, we recorded 500 sample trials with three monitors (a 21" Dell CRT, 19" Dell flat panel, and an LG 55" flat panel, see Materials and Methods of for specifications), while the Datapixx recorded an analog signal from a photodiode

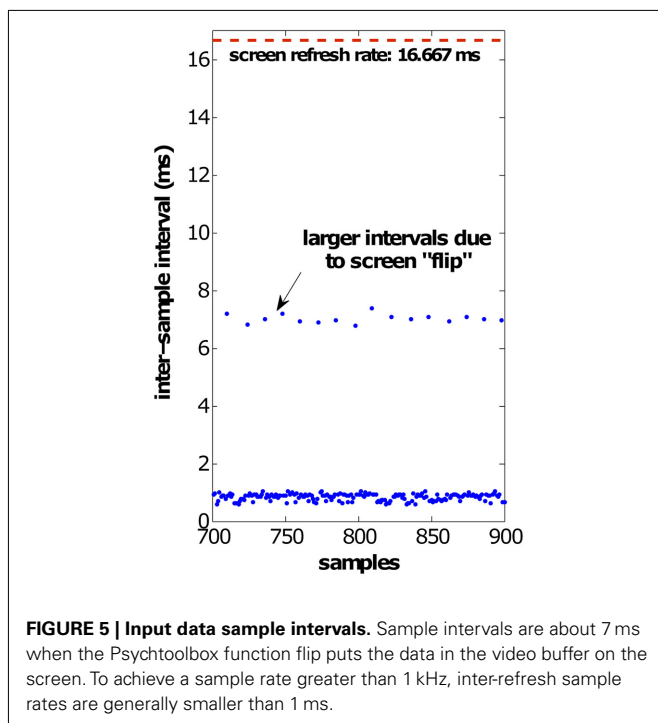
at 200 kHz. This analog signal was thresholded at the midpoint of its range and the timestamps. The results are shown in Table 2.

The configuration of PLDAPS necessitates the existence of three independent clocks: that of the stimulus Macintosh, the Plexon MAP system, and the Datapixx peripheral. The accurate synthesis of neural and behavioral timestamps would be greatly impeded if these clocks did not run at the same rate. To test this, the cumulative timing differences were compared after continuous running with the function **RunDecisionTrial** cycling through 500 trials, which took approximately 1.2 h. The MAP processor was the slowest clock, with the stimulus Macintosh computer running 0.00056% faster. For our purposes, this difference can be treated as negligible. However, the Datapixx peripheral's clock ran 0.015% faster, a difference that is much larger than between the other two clocks. So, in the course of a typical 3–4 h experiment, the timing between the two systems could be as much as 2–3 s. For this reason, special care was taken to measure both neural and behavioral information from a clock register at the start of each trial (for each respective clock). In doing so, the timing differences between the Datapixx and the MAP processor are minimized to a mean of 0.134 ms, with a SD of 2 ms and an upper limit of 1 ms. Of course this variability is dependent upon each trial running an average of 7.75 s (SD = 3.1 s) with longer trials assumed to create more variability.

DISCUSSION

The PLDAPS design goals were formed from the struggles of our lab to maintain a high-level of productivity with our current electrophysiological hardware and software. The initial idea of developing a MATLAB toolbox, instead of a stand-alone program, came from the prospect of spending weeks developing a user interface that would be obsolete when we would later decide upon a new experimental paradigm. At the same time, it became clear that many researchers in this area are well-versed in programming in MATLAB, and with Psychtoolbox in particular. Furthermore, as cognitive neuroscience expands, the number of different experimental paradigms to pursue has grown, leading to the prospect of either a dedicated programmer to code all experiments, or a flexible framework to empower each lab member as an experimental programmer while maintaining high levels of both usability and temporal precision.

The primary challenge in building such a system is the connection between the neurophysiological equipment and the experimental stimulus and control computer and software. In this case, this amounted to needing something that would connect our Plexon MAP to a Mac Pro running Psychtoolbox. Both the Plexon system (as well as other modern neurophysiological recording systems) and Psychtoolbox are well-established and in common use in many laboratories. The Datapixx box, a more recent introduction to the market, serves as an important peripheral in their integration. By allowing precise timing of inputs and outputs that were accessible by MATLAB and Psychtoolbox, it allowed development of a system in which the Mac running Psychtoolbox is the single master computer (controlling experimental state flow and generating/presenting stimuli). The Plexon MAP then records key experimental events (determined or detected by the Mac Pro) that are temporally co-registered with the neurophysiological data. The system can also pass neurophysiological data from the Plexon MAP



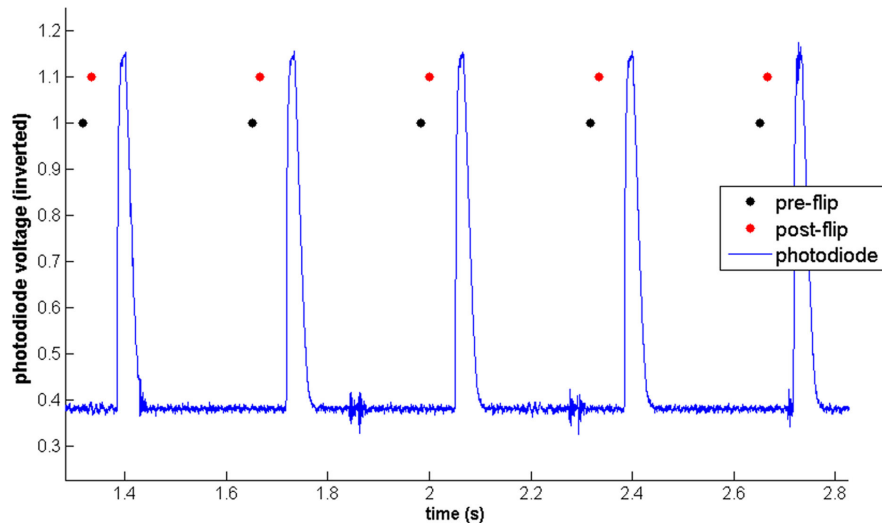


FIGURE 6 | Photodiode test. A photodiode was placed in front of a variety of monitors while every tenth screen was flashed. Analog input data (voltage) is displayed in blue as a function of time. Pre-flip and post-flip timestamps are shown in black and red, respectively.

Table 1 | DIO channels for Datapixx to Plexon communication.

Channel	usage	First pulse	Second pulse	Third pulse
1	Fixation point 1	On	Cursor in	Off
2	Fixation point 2	On	Cursor in	Off
3	Dots	On	Off	–
4	Choice	In target	Reward given	–
5	Targets	On	Off	–
6	Beginning of trial	–	–	–
7	Unused	–	–	–

Table 2 | Measured time delays between video update signal and actually screen redraw.

	LG FP 55"	Dell FP 19"	Dell CRT 21"
Mean flip-to-screen delay (ms)	52.4634	18.444	10.8201
Std flip-to-screen delay (ms)	0.0643	0.8384	0.0506
Std inter-flip interval (ms)	0.0001	0.0001	0.0001

The delay of the 55" LG monitor (our primary stimulus device) is notable, much more than the other two screens, and will be compensated for in data analysis.

back to the Mac Pro (again, via the Datapixx box), allowing the Mac the ability to “listen” to the spike train of a single neuron on-the-fly. This allows for the development of adaptive experiments, in which the next stimulus event (or the overall course of the trial or experiment) can be affected by the ongoing activity measured in the brain.

DISTINCTIONS BETWEEN PLDAPS AND OTHER SYSTEMS

The PLDAPS toolbox may be useful for many researchers, especially those who perform experiments that rely on adapting the stimuli and experimental states contingent either on an

experimental subject’s behavior, or on ongoing neurophysiological measures. Several other systems are available in this line of work, and here we briefly describe the relation of PLDAPS to these other valuable resources.

Historically, one of the most prevalent systems for awake, behaving neurophysiology is the REX system (Hays et al., 1982). Prior to developing PLDAPS, this was our lab’s primary system. REX involves a PC running the QNX operating system, executing a set of integrated c-code that controls the experimental state logic. The PC also relies on A/D and DIO cards to allow it to record co-registered spike times, and to interrogate the eye position on-the-fly. It then commands a “slave” Macintosh computer, whose sole responsibility is stimulus presentation. The REX system is remarkable in having excellent temporal precision thanks to the “real-time” nature of the QNX operating system, as well as the robustness of the underlying REX computer code. However, in our experience, modifying experiments involved a lengthy initial learning process, and many steps. In the authors’ experience, increasingly complex experiments were ultimately implemented with a variety of “hacks” that were at best inelegant, and which required great care in both documentation and use. This can be a scientifically and pedagogically unpleasant (and untenable)

situation. REX remains a valuable system for many sorts of experiments, and has supported a large number of groups and important research findings. Viewing REX as many laboratories' gold standard, we believe PLDAPS offers improved ease of use and flexibility in developing experiments and exploiting new video technology, at the relative expense of using time-tested hardware, a large set of pre-existing computer code, and perhaps a small degree of temporal precision.

Several other systems have been developed and used for various forms of neurophysiology in primates and other animals (e.g., Cortex, LabLib, EXPO, Orion; see Web References). These are important resources for the field, and PLDAPS fulfills a distinct user need. Specifically, most of these programs require learning a new set of both low-level and high-level commands for controlling experiments and formatting data. The strength of these systems is that they are mature, already have an established user base, and have been refined to perform specific types of experiments very well. However, this strength places limits on the generality of these tools. For example, several were designed primarily for the goals of researchers performing anesthetized neurophysiological recordings, and now require adaptations and elaborations to accommodate complex online behavior. PLDAPS, in contrast, was fundamentally built around a desire to use the Psychophysics Toolbox, which is inherently flexible and well-suited to behavioral experiments. Thus, PLDAPS puts virtually all the coding responsibility in the hands of each laboratory or experimenter. Depending on the goals and skill set of specific laboratories, this is either an advantage or a burden.

Finally, there are systems that were built with Psychtoolbox, such as Monkeylogic (Asaad and Eskandar, 2008). In contrast to our approach, Monkeylogic provides a large set of MATLAB scripts and functions, and is primarily focused on the behavioral components of the experiment. Thus, it provides a broad set of tools for a variety of behavioral paradigms, but is challenged by situations where monitoring of events and changing of a stimulus pattern has to be interleaved quickly (e.g., within 2–3 ms). It also does not explicitly pass data back and forth with a system like the Plexon MAP, requiring parallel and modular collection of behavioral data and neurophysiological data, and thus requires critically on *post hoc* alignment of the two types of data. PLDAPS is distinct in providing a minimalist set of tools that are focused on sub-millisecond event timing and deep integration of the behavioral and neurophysiological events.

CAVEATS AND LIMITATIONS OF PLDAPS

Given the fast timing of neurophysiological signals relative to the timing of video refreshes, a major concern in using any system (PLDAPS included) is the potential for delays, mis-registration, and temporal drift. In this paper we have attempted to clarify the operating limits of our current implementation of the PLDAPS system. In general, the speed of the CPU in a modern Macintosh Pro is not a rate-limiting factor. The limiting factors come in to play when considering how the Mac Pro communicates with the Datapixx box (a USB 2.0 peripheral) and the Plexon MAP.

The major timing caveat is that, while pre-planned events can be designed and recorded with excellent temporal accuracy

and precision (i.e., microsecond scale), “on-the-fly” events will necessarily be detected and recorded on slower time scale (i.e., millisecond), which can certainly be faster than the conventional time scale of monitor redraws. This distinction occurs because the Datapixx box can receive and pre-load planned event signals into a buffer, which can then be programmed to output the signals at microsecond scale; in contrast, unplanned on-the-fly events must instead be passed over the USB 2.0 connection (between the Mac and the Datapixx box) and processed in MATLAB, which is of course slower. However, we emphasize that in practice the Datapixx box supports microsecond accuracy for many important events, and can be used to detect and record on-the-fly events with speeds sufficient to adjust the video display within a single redraw. For many purposes, this is fast enough, but there may be exceptional conditions for which this is a fundamental temporal constraint.

Another timing issue involves clock drift over long time frames. Although this is not a factor over the course of seconds (the usual duration of trials in these experiments), we detected noticeable drift between the Datapixx clock and those of the Plexon MAP and the Mac Pro. This has two main implications. First, conventional experiments using short trials should make sure to align the Datapixx-time-stamped and Plexon-time-stamped events to the start of each trial (as opposed to, say, the start of each experimental session). Second, unconventional experiments involving long, continuous parallel recordings should contemplate correction for relative temporal drift.

Another timing-related caveat we have emphasized throughout this paper is that the PLDAPS toolbox is a minimal set of code. It is meant to serve as a set of examples of how to run simple experiments while integrating the three hardware components. Development of new experiments is left to the user, and hence, whether their particular code “runs fast” is their responsibility. It is certainly possible that large amounts of on-the-fly computational demands on the Mac can cause critical loops to be delayed relative to intended event timings. We advise performing standard timing checks (e.g., recording elapsed times between key events, checking for missed video frames) when developing new experiments using the PLDAPS tools.

POSSIBLE EXTENSIONS

A major benefit to building a system out of popular and well-maintained components is the possibility for extensions of PLDAPS's functionality. Virtually any hardware input or output device can be controlled by the Datapixx box and the Mac running MATLAB. Likewise, because the experiments themselves are designed in MATLAB and Psychophysics Toolbox, future additions and capabilities will be inherited by PLDAPS. For example, as Psychophysics Toolbox adds additional support for new video cards with additional capabilities, PLDAPS users will immediately be able to use these tools. Another intriguing possibility is the use of MATLAB's Parallel Computing Toolbox. This set of tools allows users to perform computations on multiple CPUs (or GPUs) in parallel. Given that current Mac Pros are multicore, these tools may further extend the amount of online computation possible at frame rate.

CONCLUSION

This description of the PLDAPS toolbox is meant to be helpful to the neuropsychological community on multiple levels of implementation. Many labs that perform single or multi-unit electrophysiology already use the components of PLDAPS, and can start using it as a finished system. Other labs or experimenters with different needs may want to integrate certain components into their own systems, or use principles described in PLDAPS for their own designs. At the very least, it serves as “proof of concept” of an approach to experimental software design that acknowledges the specific level of programming abilities of the typical neuroscience researcher; many are skilled enough to design their own experiments, but only with the help of a toolbox that takes care of tedious technical issues.

WEB REFERENCES

Cortex: http://www.cortex.salk.edu/documentation/VCortex11_21/VCtxUser.htm

REFERENCES

- Asaad, W. F., and Eskandar, E. N. (2008). A flexible software tool for temporally-precise behavioral control in Matlab. *J. Neurosci. Methods* 174, 245–258.
- Brainard, D. H. (1997). The psychophysics toolbox. *Spat. Vis.* 10, 437–442.
- Hays, A. V., Richmond, B. J., and Optican, L. M. (1982). A UNIX-based multiple process system for real-time data acquisition and control. *WESCON Conf. Proc.* 2, 1–10.
- Huk, A. C., and Shadlen, M. N. (2005). Neural activity in macaque parietal cortex reflects temporal integration of visual motion signals during perceptual decision making. *J. Neurosci.* 25, 10420–10436.
- Kleiner, M., Brainard, D., and Pelli, D. (2007). What's new in Psychtoolbox-3? *Perception* 36 ECVF Abstract Suppl.
- Pelli, D. G. (1997). The VideoToolbox software for visual psychophysics: transforming numbers into movies. *Spatial Vision* 10, 437–442.
- Wagner, F. (2006). *Modeling Software with Finite State Machines: A Practical Approach*. Boca Raton: Auerbach Publications.
- EXPO: <https://corevision.cns.nyu.edu/>
 Orion: <http://leelab.yale.edu/Software.html>
 PLDAPS: <http://hukdata.cps.utexas.edu/archive/PLDAPS.html>
 Liblab: <http://maunsell.med.harvard.edu/software.htmlmggma>
 REX: <http://www.nei.nih.gov/intramural/lsr.asp#software>
 PsychToolbox performance tuning: <http://psychtoolbox.org/wikka.php?wakka=FAQPerformanceTuning1>
 Plexon digital configuration: <http://www.plexon.com/assets/pdf/DigitalInput.pdf>

ACKNOWLEDGMENTS

We would like to thank Peter April and Leor Katz for their help in developing PLDAPS. We appreciate Thad Czuba's assistance with Figure 1. This work was supported by the following grants from the US National Institutes of Health/National Eye Institute: R01-EY017366 (US NIH) to Alexander C. Huk, and R01-EY020592 to Alexander C. Huk, Lawrence Cormack, and Adam Kohn.

Citation: Eastman KM and Huk AC (2012) PLDAPS: a hardware architecture and software toolbox for neurophysiology requiring complex visual stimuli and online behavioral control. *Front. Neuroinform.* 6:1. doi: 10.3389/fninf.2012.00001

Copyright © 2012 Eastman and Huk. This is an open-access article distributed under the terms of the Creative Commons Attribution Non Commercial License, which permits non-commercial use, distribution, and reproduction in other forums, provided the original authors and source are credited.

Received: 04 August 2011; accepted: 03 January 2012; published online: 31 January 2012.