Behavior Research Methods
2010, 42 (1), 212-225
doi:10.3758/BRM.42.1.212

RTbox: A device for highly accurate response time measurements

Xiangrui Li
University of Southern California, Los Angeles, California

Zhen Liang
University of Science and Technology of China, Hefei, China
and Anhui Medical University, Hefei, China

Mario Kleiner
Max Planck Institute for Biological Cybernetics, Tübingen, Germany

and

Zhong-Lin Lu
University of Southern California, Los Angeles, California

Although computer keyboards and mice are frequently used in measuring response times (RTs), the accuracy of these measurements is quite low. Specialized RT collection devices must be used to obtain more accurate measurements. However, all the existing devices have some shortcomings. We have developed and implemented a new, commercially available device, the RTbox, for highly accurate RT measurements. The RTbox has its own microprocessor and high-resolution clock. It can record the identities and timing of button events with high accuracy, unaffected by potential timing uncertainty or biases during data transmission and processing in the host computer. It stores button events until the host computer chooses to retrieve them. The asynchronous storage greatly simplifies the design of user programs. The RTbox can also receive and record external signals as triggers and can measure RTs with respect to external events. The internal clock of the RTbox can be synchronized with the computer clock, so the device can be used without external triggers. A simple USB connection is sufficient to integrate the RTbox with any standard computer and operating system.

In behavioral studies, response times (RTs) provide valuable measures of human performance (Jastrow, 1890; Luce, 1991). Defined as the lapse of time between stimulus or task onset and a subject's response, RTs have been measured in a variety of tasks, ranging from simple visual and auditory detection (Arieh & Marks, 2008), choice reaction (Brown, Marley, Donkin, & Heathcote, 2008), and object recognition (Lu, Morrison, Hummel, & Holyoak, 2006) tasks to more complex lexical decision tasks (Yap, Balota, Cortese, & Watson, 2006). RT is also a key component in speed–accuracy trade-off paradigms (Dosher, 1976; Ratcliff & Smith, 2004). In this article, we describe the design, implementation, and test results of a new device, the *RTbox*, for accurate RT measurements in behavioral experiments. We focus on the most commonly used response mode, button responses.

Accurate measurements of RTs require recording of two time stamps, the onset time of a stimulus or task and the time of a subject's response, with millisecond accuracy. It would be ideal that both time stamps are represented in the same time base—that is, according to the same clock. They should also be reliable, free of systematic bias and excessive measurement noise, and not affected by any timing jitter caused by standard computer hardware and operating systems. For an RT measurement device to be widely applicable, it must be (1) compatible with the standard, widely used computer hardware, such as Intel PC-compatible computers and Apple Macintosh computers; (2) compatible with the commonly used operating systems, including Microsoft Windows, Apple Mac OS X, and GNU/Linux; (3) easy to use and control from stimulus presentation software; and (4) easy to set up (only a minimal amount of work is required to integrate the device into common experimental setups). To reduce the impact of other component devices in experimental setups on RT measurements, it is also essential that the device can collect a subject's responses asynchronously: It should be capable of storing the timing of all the response events until it is convenient for the user software to retrieve them. The ability to detect and time stamp signals from external devices—for example, transistor–transistor logic trigger signals from a stimulus or data acquisition

Z.-L. Lu, zhonglin@usc.edu

device—is also a very useful feature. Before describing our RTbox, we review a few commonly used RT collection methods and devices.

Computer keyboards and mice are widely used for RT measurements. These devices are cheap, do not require any special hardware setup or programming, and naturally work with any computer hardware, operating system, and standard software toolkit. However, the use of a keyboard and mouse often leads to huge variations and biases in RT measurements. Figure 1 shows a typical sequence of events involved in obtaining a buttonpress response with a regular computer keyboard. There are several sources of timing error in keyboard responses.

1. Mechanical lag. Typical keyboard and mouse buttons are implemented as electrical contacts that close and open an electronic circuit when the subject presses and releases a key or button. A keyboard encoder chip is used to register each closing of the electric circuit as a key-/buttonpress. Depending on the mechanics of the keys/buttons, there is some delay between key/button press/release and circuit switch on/off. This kind of delay cannot be eliminated totally, but some keyboards and mice may introduce unknown long delays.

2. Debouncing. Due to mechanical imperfections, button bouncing—that is, an electrical contact opening and closing the circuit multiple times in quick succession for a single key-/buttonpress—often happens. To eliminate the errors caused by button bouncing, the encoder chip performs *debouncing*—treating switch-on and switch-off events within a certain time window as a single event. Although the length of the time window and the exact implementation of the debouncing algorithm vary across keyboard models and manufacturers, a 20-msec delay due to debouncing is not uncommon. This introduces a systematic delay but does not affect relative RT measurements.

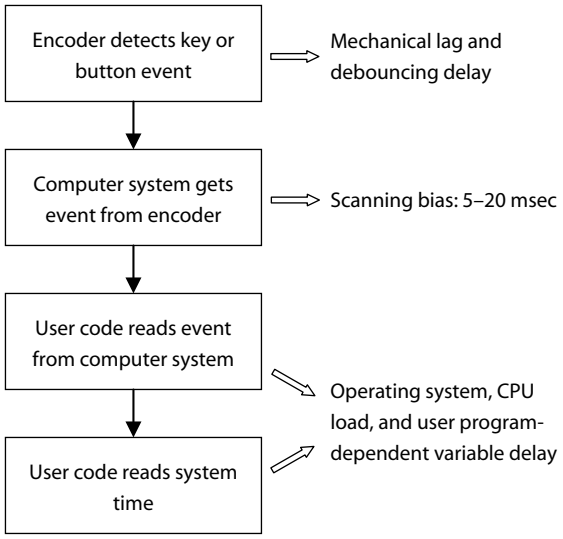

Figure 1. A diagram showing the steps involved in measuring response times with a regular computer keyboard/mouse. Each step may introduce some timing error, as summarized on the right side of the figure.

3. Scanning. To save manufacturing costs, there is not a dedicated electric connection from each key on a keyboard to an input of the encoder chip. Instead, the electrical contacts are arranged in a matrix form, with the switches placed at the intersections of the rows and columns in the matrix. A keyboard with up to 128 keys typically uses 24 connections on the chip and an 8×16 matrix. Pressing a key closes the electric circuit in one row and one column of the matrix. The keyboard encoder sequentially and periodically scans the connections in all the columns and rows and uses the state of the matrix to identify the key/button pressed/released. Keys connected to different rows and columns are checked at different times in the cycle. Depending on the keyboard encoder, the scan cycle typically takes 5–20 msec and may introduce a 5- to 20-msec timing bias among keys (Shimizu, 2002) if multiple keys at different rows/columns of the matrix are used in an experiment. A mouse typically has only one to three buttons, so it is not vulnerable to scanning bias. However, some mice may have up to a 70-msec delay and variations due to other design issues (Plant, Hammond, & Whitehouse, 2003).

4. Polling. Mice and keyboards are usually connected to their host computers via the USB using the USB–HID protocol—an inexpensive, flexible, and standardized communication protocol. USB–HID, like all USB protocols, is based on periodic polling. The host computer periodically queries input devices for state changes—for example, mouse movements or keyboard buttonpresses. Detection of any keyboard or mouse event is quantized in time to the boundary between successive polling cycles. The standard polling interval for mice and keyboards is typically fixed at 8 msec. This introduces an additional uncertainty of 8 msec, independently of the type of the host computer, operating system, or software toolkit.

5. Event handling. Additional uncertain delays could arise due to the variability in the latency between event detection by the USB controller and the handling of the event by the operating system, as well as event registration by the user program. These delays depend mostly on the operating system, the user software toolkit, and program, as well as the computational load on the host computer's central processing unit (CPU). It can vary from being negligibly short to lasting hundreds of milliseconds or even longer. General-purpose operating systems, not designed for reliable real-time control, do not provide any corrective mechanisms against any of these software-induced variable delays. In some software toolkits, the user program needs to control the stimulus, such as refreshing the visual display in each video frame, and, therefore, may increase the response check interval, thus making the time measurements less accurate.

Generally speaking, regular computer keyboards and mice are optimized for daily use and low manufacturing costs, not for scientific experiments with highly accurate timing, which is also the case for the standard USB or any general-purpose operating systems. Putting together the various sources of timing errors in keyboard responses can add up to create delays and uncertainties of between 20 and 70 msec, comparable to or greater than the vari-

ability of human RTs, which is typically about 20 msec in simple detection tasks. Perhaps more important, variability in human RTs is normally distributed (Luce, 1991), but some of the timing variability in keyboard/mouse responses is biased. The bias may make comparisons of RTs between multiple conditions invalid. In situations in which the possible RT difference is small, the additional variability from the measurement device may make the difference undetectable. A dedicated solution, other than keyboard and mouse, is needed for high-accuracy RT measurements.

There exist several commercial and custom-made RT measurement devices, although detailed descriptions of the design principles for most of them are not publicly available. As an example of a do-it-yourself solution, Stewart (2006) described an inexpensive button box with millisecond accuracy under the Linux operating system. Although his solution can also be implemented in the Windows operating system, its requirement of a parallel port can no longer be satisfied with most current computer hardware. A PCI card with a parallel port must be added to the computer.

Some commercial RT devices are essentially improved computer keyboards. The advantage is that they work with minimal setup and across all software and hardware platforms. For example, the button box and keyboard from Empirisoft Corporation (www.empirisoft.com/directin.aspx) utilize high-quality keys with good mechanical properties in combination with a specially designed keyboard encoder, but with a standard USB connection and the standard keyboard device drivers of the host operating systems. The keyboard encoder employs a debouncing method optimized for low timing bias and high-speed scanning to reduce the delay between a keypress and its detection by the host computer to about 1 msec. If a polling interval of 1 msec can be achieved, the device is capable of transmitting a key event to the host computer with a timing uncertainty of 1–3 msec. However, the 1-msec polling interval is not widely available for USB 1.1 compliant devices. The typical polling interval is 8 msec.

Many button boxes use a native serial port connection to transmit buttonpress events to the host computer. Upon detecting a buttonpress event, a byte is sent to the host computer with a delay that is typically less than 1 msec. The host computer waits or polls for receipt of such single event bytes and calculates an event time stamp on receipt. Examples are the response box from Engineering Solutions, Inc. (www.response-box.com/tools.shtml) and the PST Serial Response Box from Psychology Software Tools Inc. (www.pstnet.com/products/SRBOX/default.htm). Although the manufacturers claim that these devices are capable of reducing the mechanical lag, debouncing time, and scanning bias, standard serial ports are not commonly available on commercial computers. Serial ports emulated over USB connections usually have a longer latency than do native serial ports. Both emulated keyboards and serial port response boxes are also vulnerable to timing uncertainty due to processing delays of the operating system. In addition, the response collection software must be specifically written to perform button events polling with high precision and reliability, which is not an easy task on general-purpose multitask operating systems, where any piece of regular user software can be interrupted or delayed in its execution by the operating system for an unbounded amount of time.

In order to reduce timing uncertainty caused by processing delays within the host computer system and experiment toolkit and to simplify the implementation of user programs, a response device should have its own clock and microprocessor to record button event times independently of activities on the host computer.

The PsyScope Button Box (http://psyscope.psy.cmu.edu/bbox/index.html) is designed to have its own clock, but according to its manual, its clock mechanism has never worked properly, so this feature is not usable. The RB Series Response Pads from Cedrus Corporation (www.cedrus.com/support/rb_series) is the closest to our definition of a suitable response box. The RB device has its own clock and microprocessor for independent event detection, time stamp assignment, and buffering with millisecond accuracy. It can be connected to a standard USB port and can communicate with experimental software via a serial port protocol emulated on top of the USB connection. This allows any experimental software with support for serial ports to communicate with the devices. The device can receive external TTL signals indicating stimulus onsets, and there are different button designs for different purposes. This is a well-designed device for accurate RT measurement. However, one of the authors (M.K.) found that it is not easy to accurately synchronize the clock of the box with the computer clock, making it difficult to compare stimulus onset time stamps created by the host computer software and response time stamps created by the box.

We have designed and implemented a device, the RTbox, for highly accurate RT measurements, to satisfy all the considerations of an ideal response device described in the beginning of this section. The RTbox has its own microprocessor and high-resolution clock, so it can record the identities and timing of button events with high accuracy, unaffected by potential timing uncertainty or biases during data transmission and processing in the host computer. It stores button events until the host computer retrieves them. The asynchronous storage greatly simplifies the design of user programs. The RTbox can also receive and record external signals as triggers, measure RTs with respect to external events, and synchronize other equipment with the device. The internal clock of the RTbox can be synchronized with the computer clock. A simple USB connection is sufficient to integrate the RTbox with any standard computer and operating system.

DESIGN

The RTbox is a USB 1.1 and 2.0 compatible device with four response buttons. In the simplest setup, the user needs only to plug it into a USB port. The RTbox also has two external input ports for receiving TTL and light triggers.

Compatibility and Choice of Interface

To design a device that would work with both PC and Apple Macintosh hardware, both desktop and portable computers, and all the popular operating systems, its interface can be either USB or Ethernet ports. However, most computers have only one Ethernet port; some recent portable computers do not have any Ethernet ports. USB ports are available on all commercial computers as the preferred choice for connecting peripheral devices, and the number of USB ports on a computer can be easily extended with a USB hub, if needed. Thus, we chose USB as the interface for the RTbox.

Implementing custom communication protocols as native USB protocols is complex and would require development of dedicated low-level device drivers for each supported operating system, as well as development of special interface routines for each supported experimental software toolkit. To avoid this overhead, we adopted a simple and popular solution by implementing a standard serial bus protocol on top of the USB protocol by use of a USB-to-serial converter chip. The serial port data transmission protocol is well supported by all the commonly used operating systems and most experimental toolkits. The disadvantage of the serial data transmission protocol is its slow speed, but this is not an issue, since the amount of data is only several bytes for an event and transfer of event data to the host computer is not time critical for an asynchronous device.

In the RTbox, we use the FT232BM USB-to-serial converter chip from Future Technology Devices International Limited (www.ftdichip.com). This chip is supported by Microsoft Windows, Apple Mac OS X, and GNU/Linux operating system families and is compatible with all computers with a USB 1.1 or 2.0 host controller.

Implementation of Event Detection, Time Stamping, and Recording

To eliminate the unpredictable and variable delays during event data transfer to the host computer and processing of the data by the operating system and user programs, an ATmega16L microprocessor from Atmel (www.atmel.com), with a clock temporal resolution of 8.68 μsec, is incorporated into the RTbox to time stamp button events, independently of the host computer. The design is similar to those using a second slave computer (De Clercq, Crombez, Buysse, & Roeyers, 2003) or an external timer (Chambers & Brown, 2003; McKinney, MacCormac, & Welsh-Bohmer, 1999) for time stamping.

In the RTbox, the microprocessor is connected to the USB-to-serial converter chip for serial port communication with the host computer and, via digital input interface electronics, to the buttons and the two external input ports (Figure 2). The firmware on the microprocessor periodically monitors the states of the TTL port, light port, serial port, and four response buttons at an interval shorter than 0.1 msec. If a state change is detected on any of them, the firmware reads the current clock time of the microprocessor, assigns it as the time stamp to the event, and writes the corresponding event code and time stamp to the serial output port. The complete event packet, 7 bytes

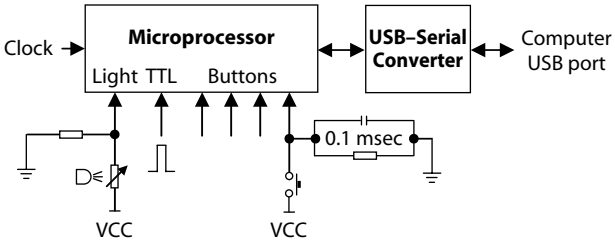

Figure 2. A diagram illustrating the basic hardware components of the response box. An external oscillator (7.3728 MHz) is used in the microprocessor and is scaled down by 64 to perform time stamping. A resistor–capacitor circuit (time constant, 0.1 msec; shown for only one of the four buttons) is used for buttonpress and release debouncing. The microprocessor can detect TTL pulses and light signals generated by a photodiode (shown as an adjustable resistor). The serial port of the microprocessor performs serial communication with the host computer through the USB-to-serial converter chip.

for each event, is stored by the serial buffer of the host computer. The user program is free to retrieve the data whenever it is convenient—for example, at the end of each experimental trial.

Selective Event Detection

The RTbox is also designed to detect both buttonpress and release events. Some researchers prefer to measure RTs using button releases, instead of buttonpresses, because the mechanical lag in button releases is shorter for some buttons. In a given experiment, the user is usually interested in only a subset of events. Detection of specific events can be enabled and disabled by a simple serial command from the user program. The *disable* mode can be used to reduce the amount of data if no event detection is necessary during certain periods of an experiment.

Button Events and Debouncing

In the RTbox, the button-scanning interval is less than 0.1 msec if the microprocessor is not writing data. For button debouncing, we use a combined hardware and software approach. In the hardware, we use a 0.1-msec resistor–capacitor circuit (Figure 2) to filter out quick button on/off switches. In the firmware, we treat three consecutive identical results of the button-scanning loop as a valid button event. Since our scanning interval and debouncing time are very short, their time delay is negligible. This is ideal for accurate RT measurements, but it is still too short to avoid button bouncing. However, unlike problematic multiple events for keyboards and mice, it is not a problem for the RTbox. When repeated events are detected within a given time window, such as 50 msec, we can simply use the first buttonpress event or the last button release event in the host computer driver, without introducing additional delay or uncertainty.

External Trigger Inputs

The RTbox can receive external triggers that mark stimulus onsets to obtain RT measurements completely independently of the host computer and its software. Used in this way, the accuracy of RT measurements depends

mostly on the timing accuracy of the external trigger. The RTbox can receive two types of external triggers: standard TTL pulses and light signals generated by a photodiode. The TTL triggers can be generated by a wide range of devices. A photodiode attached to an unused region of the computer monitor can also generate trigger events by detecting luminance changes synchronized to the onsets of visual stimuli. By recording both trigger and button-press events in the RTbox, accurate RTs can be calculated by subtraction of the trigger time stamp from the button time stamp. This feature is especially useful when it is implemented with presentation software that is incapable of computing exact stimulus onset time stamps.

In the RTbox, the detection of a light trigger event automatically disables detection of further light triggers until the host computer enables it again with a special command. Cathode ray tube (CRT) monitors, which are widely used in vision experiments, are impulse type displays: Static images are not displayed continuously on the screen but are redrawn at every video refresh cycle (Brainard, Pelli, & Robson, 2002). Although the refresh is not perceived by humans, because of the relative slow temporal modulation transfer function of the visual system, photodiodes can detect the luminance change at each refresh cycle and cause the response box to record large numbers of redundant light events for a single static stimulus. The autodisable feature prevents this unwanted effect.

Another important use of external triggers is for calibrating experimental setups. External triggers can be used to mimic button responses of an ideal subject with no response delay or variability, so it is extremely useful for testing the timing accuracy of experimental setups and the timing of experimental stimuli.

Synchronization of the Clocks

In some experimental setups, it is not possible or convenient to generate accurate external triggers for the RTbox to use. User programs can often acquire or calculate time stamps for stimulus onsets on the basis of time measured with the clock of the host computer. The complication is that the event time stamps received from the RTbox must be converted and expressed in the time of the host computer. The clocks of the RTbox and the host computer must be synchronized.

The computer clock usually measures elapsed time since system startup or since a fixed reference point in the past, whereas the clock of the RTbox measures elapsed time since it has been powered up. We synchronize the two clocks by measuring the offset between the two clocks. Ideally, the offset should be a constant. The RTbox is designed to accept a special trigger from the host computer via the USB–serial port. The computer sends out the trigger at t_{send} measured with its own clock. When the RTbox receives this trigger, it sends its current time, $t_{receive}$, to the host computer. We use this serial trigger event to measure the offset between the two clocks:

$$t_{diff} = t_{send} - t_{receive}. \quad (1)$$

Any time stamp from the RTbox can be converted and expressed in the computer time by adding this offset to it:

$$t_{computer} = t_{device} + t_{diff}. \quad (2)$$

The accuracy of t_{diff} is crucial for accurate mapping of RTbox time stamps to the host computer time. Figure 3 shows the steps of clock synchronization and possible timing errors each step may introduce. In Steps 1 and 2, the host computer writes the trigger to the serial port and acquires the time it sends out the trigger t_{send}. The serial data transmission between Steps 2 and 3 takes time, but the time is less than 0.1 msec to transmit a single trigger byte at the baud rate of 115,200 bits/sec. The detection of the trigger arrival at the microprocessor (Step 3) can be delayed by up to one scanning interval, which is shorter than 0.01 msec when the microprocessor is idle. The device takes about 0.06 msec to read the $t_{receive}$ time stamp from its clock (Step 4). As we can see, the possible error of $t_{receive}$ is very small and relatively stable. Data transmission of $t_{receive}$ from the RTbox to the host computer may take up to 16 msec, but because this happens after all the time stamps have been acquired, it will not affect accuracy of clock synchronization. The limiting factor in clock synchronization is the accuracy of t_{send}.

The delay between Steps 1 and 2 may vary from computer to computer and from time to time, because it depends on the host computer CPU load and the operation of the USB controller. In fact, we do not have a direct way to get t_{send}. Instead, in Step 1, we take a time stamp, t_{pre}, of the computer clock right before submitting the write operation. Then we wait until the operating system signals the completion of the operation and take another time stamp, t_{post}. We have verified that, in all the common operating systems, t_{post} is after the write completion and a USB duty

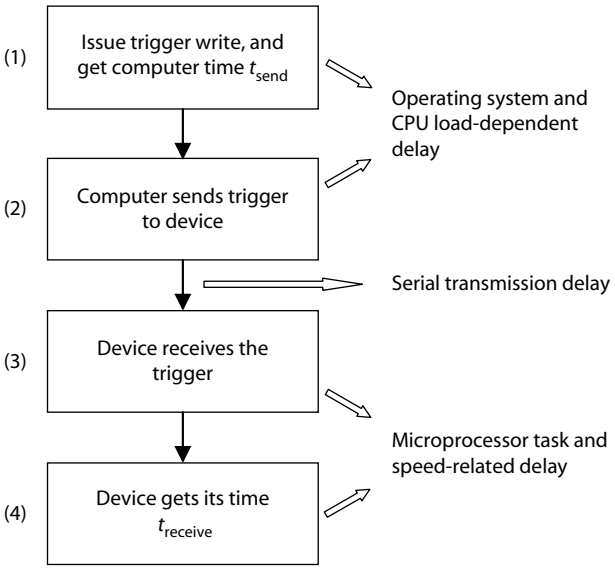

Figure 3. Steps for clock synchronization over the USB–serial link and possible time errors. The microprocessor is single-task oriented, so its delays in Steps 3 and 4 are very short and very stable. The serial data transmission takes a very short time for a single-byte trigger. The time difference between the receiving time stamp of the device ($t_{receive}$) and t_{send} is used to convert the RTbox time into computer time. The details about how to get a reliable t_{send} are discussed in the text.

cycle, which is 1 msec for most USB controllers (www.usb.org/developers/docs). Thus, we can be certain that the correct t_{send} must be between t_{pre} and t_{post}. Since both t_{pre} and t_{post} can be affected by the computer load and user program, we repeat the measurement many times and get three time stamps—t_{pre}, t_{post}, and $t_{receive}$—for each repeat. Then we have three different methods to select the best estimate of t_{send}.

Method 1. Because $t_{receive}$ is reliable, the t_{pre} associated with the minimum $t_{receive} - t_{pre}$ is the t_{pre} closest to the write operation and, therefore, the best estimate of t_{send}. In other words, we can use $\max(t_{pre} - t_{receive})$ among the repeats as the estimation of t_{diff} in Equation 1.

Method 2. The t_{post} associated with the minimum $t_{post} - t_{receive}$ gives the t_{post} that is closest to the write operation and, therefore, the best estimate of t_{send}. This means that we can use $\min(t_{post} - t_{receive})$ as the estimation of t_{diff}. Because t_{post} can be obtained only after a USB duty cycle, a random time jitter from 0 to 1 msec is inserted between repeated measurements to reduce the impact of the USB duty cycle. The random jitter allows us to obtain a distribution of $t_{post} - t_{receive}$ from which to select the best t_{post}.

Method 3. The random jitter also allows us to obtain a distribution of $t_{post} - t_{pre}$. The minimum of this distribution provides the minimum time interval that contains t_{send}. When this time window is small enough (~0.2 msec), we can use the average of the corresponding t_{pre} and t_{post} as the estimate of t_{send}.

We also take some extra care in measuring t_{send}. First, real-time priority scheduling of the write and receive operations from the operating system should be used during clock synchronization to minimize the probability and duration of potential interruptions by other computer processes. Second, detection of button and other events on the RTbox is disabled to reduce the load on its microprocessor and the USB system. The difference between t_{post} and t_{pre} provides the upper bound accuracy for t_{send}. If this difference is too big, such as greater than 2 msec, the synchronization is unreliable. This typically indicates that the host computer is overloaded, and other time stamps from the host computer are probably also inaccurate. When this happens, the user should reduce the processing load of the host computer.

Using the RTbox

We have written a driver to control the RTbox in MATLAB, based on Psychtoolbox-3 extensions (http://psychtoolbox.org/). The driver is compatible with all computer systems supported by Psychtoolbox, including PCs running either Windows or Linux operating system and Macintosh computers. Once the RTbox is connected to the host computer via a USB connection, the driver can automatically detect it and control/execute all its functionalities. The link to download the driver code can be found in the Appendix. Users of other toolkits can use the code as an example in developing their own drivers.

Figure 4 illustrates the steps involved in measuring RTs using the RTbox and its MATLAB driver. In the first mode, no external trigger is used (Figure 4A). After computing the stimulus is finished in Step 1, a function call to the driver in Step 2 clears the serial buffer to remove any possible irrelevant button events from the subject during the nonresponse period of the previous trial. This function call also performs clock synchronization to get t_{diff} (Equation 1). Step 3 is to issue the stimulus and return its onset time (t_{onset}) on the basis of the computer clock. In Step 4, subject response is read from the RTbox: which button is pressed/released at what time. Normally, a response period is specified in this step because an invalid long RT is often not useful. Because the setup can buffer hundreds of events, it is not critical that we read the response from the RTbox immediately. One can optionally prepare for stimulus of the next trial before Step 4. The returned button time (t_{button}) is represented in the host computer time, converted from device time by the driver on the basis of clock synchronization in Step 2, using Equation 2. Step 5 calculates the RT from the two time stamps:

$$t_{resp} = t_{button} - t_{onset}. \quad (3)$$

Normally, in Step 6, the user can check whether the subject had missed the trial or made multiple responses and can also apply some criteria to check whether the response is valid. For example, if the RT is too short or even negative, one would treat it as an invalid response. If the response is valid, the result is recorded. Then the procedure is iterated for the execution of the next trial.

If external triggers marking stimulus onsets are available, either as TTL pulses or light signals, the RTbox can be used in another mode (Figure 4B) in which clock synchronization is not necessary. In Step 3, instead of returning the stimulus onset time, a trigger signal is sent to the RTbox by the stimulus generator. In Step 4, in addition to the button event and time (t_{button}), the trigger event and time ($t_{trigger}$) will also be read from the RTbox. Here, the event times do not need to be converted into the computer time, since we will compute their time difference in Step 5. Actually, our code can return the RT (t_{resp}) directly, so Steps 4 and 5 are actually combined into a single function call.

The Appendix also contains some example MATLAB programs based on Psychtoolbox for measuring RTs with and without external triggers.

RESULTS

We performed several tests to verify the accuracy of the RTbox, both with and without external triggers. Some of the tests can also be used to calibrate RT measurements.

Measuring RTs With External Triggers

We used the RTbox to measure the RTs of a human subject to a square flashed in the center of the computer screen (Figure 5D), with either TTL or light triggers indicating its onsets, on three computer systems: Microsoft Windows XP (Win), Mac OS X (Mac), and OpenSUSE Linux (Lnx). The subject's task was to press a button on the RTbox as soon as he saw the square. The TTL triggers were generated by a Video Switcher (Li, Lu, Xu, Jin, &

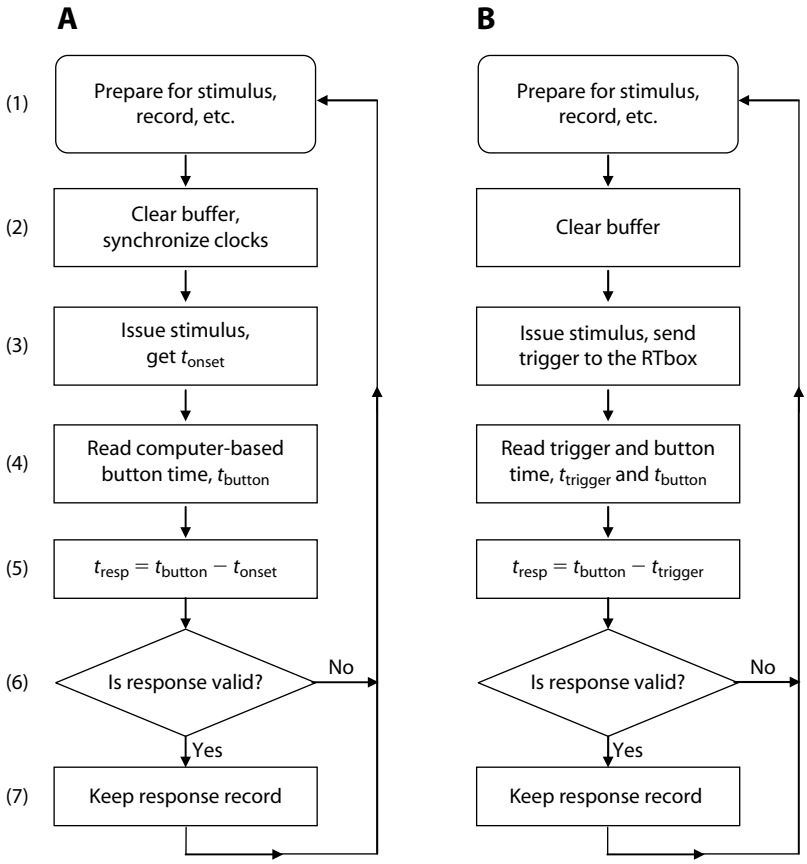

Figure 4. Flowchart of a typical response time (RT) experiment using the RTbox. (A) Steps to measure RT without an external trigger. The button time (t_{button}) at Step 4 is computed from the RTbox time and the clock difference measured in Step 2. (B) Steps to measure RTs using external triggers.

Zhou, 2003). Twenty trials were measured in each trigger mode and on each of the three computer systems.

The bold bars in Figures 5A and 5B indicate the median RTs from 20 trials, and each thin bar represents a single trial. The median RT ranges from 150 to 180 msec. The RTs are very similar across the two different trigger modes and all three computer systems ($p > .05$, Kruskal–Wallis test; the same for all the tests, unless noted otherwise).

Due to the large variation of human RTs, it is hard to tell, from Figures 5A and 5B, how accurate the device is. In Figure 5C, we measured the "RT" of an ideal observer, simulated using the photodiode. In this setup, TTL pulses served as the external triggers. The light signals served as button responses; the photodiode was used to detect the square on the computer screen, and its responses were used to simulate button responses. The result reflects the time difference between the TTL and light triggers. As is shown in Figure 5C, the variability among 100 repeats is very small: The maximum variability is only 26 μsec, which equals to three quantizations of the clock in the RTbox. This extremely small variability illustrates the high precision of the RTbox. The median RT for all three computer systems is 0.521 msec. The time for different systems are the same because we used the same CRT with identical settings (resolution of 1,024 $\times$ 768 pixels, refresh rate of 85 Hz) and the same light sensor location. The nonzero RTs of the ideal observer resulted from the experimental setup: The TTL triggers from the Video Switcher signified the first scan line of the stimulus square on the CRT, and the photodiode was placed roughly at the center of the square (100 $\times$ 100 pixels). With the monitor setting we used, 50 scan lines take about 0.54 msec, which accounts for the nonzero time difference between light and TTL. The capability of detecting such a small time difference further verifies the precision of the RTbox.

Accuracy of Clock Synchronization

To use the RTbox without external triggers, we must obtain accurate clock synchronization. Figure 6 shows synchronization results on three computer systems. We measured t_{diff} (Equation 1) once every 10 sec for a total of 30 min on each system. To verify the clock synchronization methods described in the Synchronization of the Clocks section, each t_{diff} was selected from 100 repeats of the write operation, which took about 0.5 sec to run.

The three plots in Figure 6A show t_{diff} estimated with three different methods on a Windows system. For visualization purpose, the t_{diff}s are plotted relative to the first

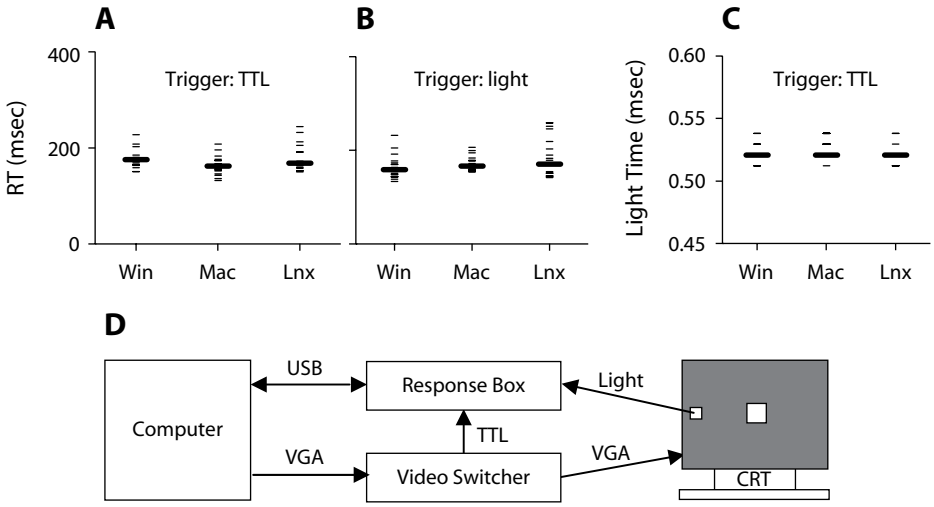

Figure 5. Response times (RTs) to a square flashed in the center of the monitor, with external triggers, on three computer systems: Microsoft Windows XP (Win), Mac OS X (Mac) and OpenSUSE Linux (Lnx). Each thinner bar represents a trial. The bold bars indicate median RTs. (A) RTs measured using TTL pulse triggers from a Video Switcher. (B) RTs measured using the light trigger at the same horizontal screen location. (C) The time of the light signals relative to TTL triggers. (D) A diagram showing the hardware setup of the tests.

measurement. Clearly, t_{diff} is not a constant but shows a linear drift (absolute values of Pearson's linear correlation coefficients greater than .9999999) over time for all the three methods. Within 30 min, the drift can be as high as 240 msec, because the host computer clock and the RTbox clock do not operate at the same speed, usually caused by imperfections in the manufacturing of the quartz oscillator crystals within the clock hardware. The slope of linear drift is -1.37×10^{-4} sec/sec, and the slope differences among the three methods are extremely small, only on the fifth significant digit. After correcting the clock speed difference by removing the linear drift, the variation of t_{diff} over time is shown in Figure 6B. With clock speed corrected, the clock difference is almost a constant over time. The nonlinear component is very small, within 0.1 msec for all three methods. This nonlinear drift is probably due to environmental effects—for example, fluctuations in power supply and operating temperature—on the clocks.

Figures 6C and 6D show the variation of t_{diff} after clock speed correction for the Mac and Lnx systems. The slopes from the three t_{diff} selection methods are the same for each system (-1.75×10^{-4} sec/sec for Mac; -7.95×10^{-5} sec/sec for Lnx). The Mac system does not show visible nonlinear drift. Although we see some outliers on the Lnx system, the maximum variation range is less than 0.2 msec, which is good enough for most RT experiments.

In an experiment, the linear and nonlinear drifts of the clocks are normally not a problem if we perform clock synchronization right before the onset of each stimulus (Step 2 in Figure 4A). The total accumulated drift and, therefore, the measurement error are very small within a typical response interval (less than 0.2 msec within a second; see the slopes of the linear drift on the three systems). Clock synchronization normally takes less than 200 msec, so it is not a major inconvenience to perform it just before each trial. It is also very easy to correct the clock speed. Our driver software can also take some samples in a period of time, such as 30 sec, compute the slope of drift, and remove the linear drift. Suppose that the slope of the drift is m; the driver can be used to remove the drift by computing the corrected device time:

$$t_{corrected} = t_{uncorrected}(1 + m), \quad (4)$$

where $t_{uncorrected}$ is the uncorrected device time based on the device clock. The driver software can automatically apply the correction to all measured RTbox times, such as $t_{receive}$ and t_{device} in Equations 1 and 2.

Although all three synchronization methods are very good, each of them has some advantages and disadvantages. From Figure 6B through 6D, Method 1 [$t_{diff} = \max(t_{pre} - t_{receive})$] gives the smallest variation over time on all three tested systems. This is because, on a computer system with a normal CPU load, the write operation of the trigger byte is very close to t_{pre} most of the time, so it is easy to get a t_{pre} that is very close to the real write operation. Actually, 20 repeats in 0.2 sec are sufficient to obtain good clock synchronization. However, this method is based on time stamps from two clocks: t_{pre} from the host computer clock and $t_{receive}$ from the RTbox clock. So, it relies on the assumption that the RTbox time stamp $t_{receive}$ is very reliable. Although this is true in most circumstances, the linear drift (Figure 6A) due to clock speed difference can affect repeated measurements of $t_{pre} - t_{receive}$ and, therefore, the selection of the best sample from them. Figure 6E shows the variability of t_{diff} with and without correcting the clock speed on the Mac system. Although the difference is small, clearly the best t_{diff}s are different. The range of variation is 0.040 and 0.021 msec before

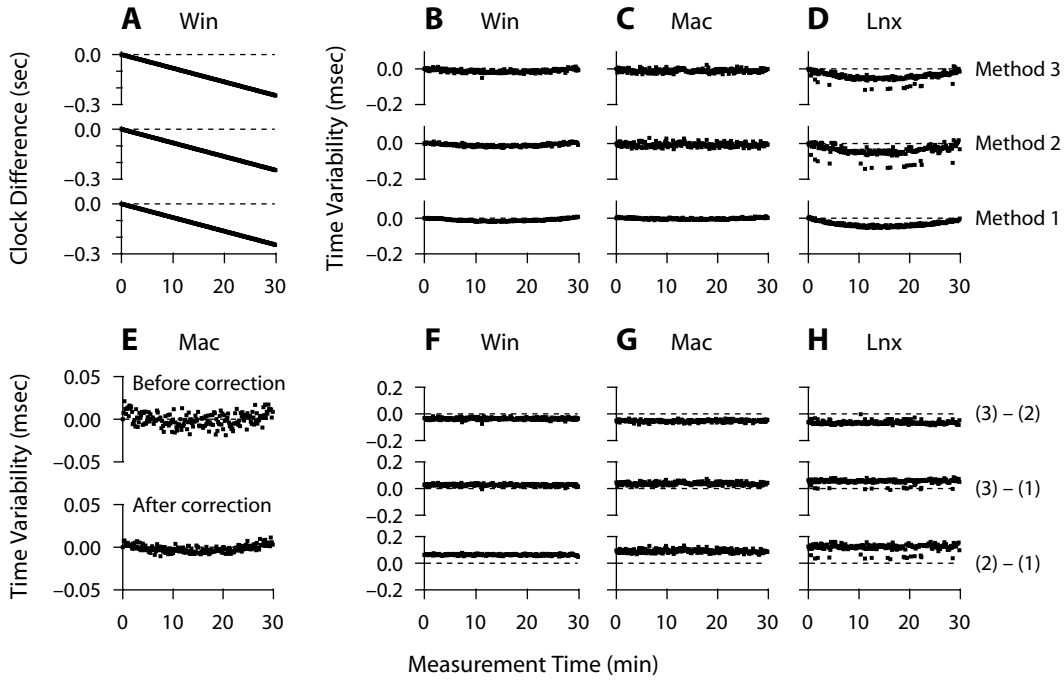

Figure 6. Clock synchronization results on the three computer systems. (A) Clock time difference between the host computer and the RTbox over time on a Win system. The test was repeated 180 times in 30 min, once every 10 sec. For visualization purposes, the differences are relative to the first repeat. The horizontal axis represents the time of the measurements. (B, C, D) Clock difference variability from three different estimating procedures, on the three systems. (E) Clock difference variability on the Mac system before and after clock speed correction. (F, G, H) Possible systematic differences among different synchronization methods on the three systems.

and after clock speed correction, respectively. With an enlarged vertical scale, we can also see a very small nonlinear drift that is not visible in Figure 6C.

Method 3 [selection based on $\min(t_{post} - t_{pre})$] uses only the time stamps from the computer clock to choose the best sample, so the clock speed difference does not affect the sample selection. Because the method relies on the random time jitter between repeats, a relatively large number of repeats (100, e.g.) are necessary to increase the possibility that at least one of the samples is very close to the end of a USB duty cycle. Method 2 [$t_{diff} = \min(t_{post} - t_{receive})$] relies on both the random waiting jitter and the accuracy of device time stamps, so it is inferior to the other two methods, although it could get the best sample whose write operation is delayed from t_{pre} but happens to be close to the end of a USB duty cycle.

Figures 6F, 6G, and 6H show the differences of t_{diff} (Equation 1) estimated with the three different methods on the three computer systems. If all three methods were reliable, any difference between them would be very small. The t_{diff} from Method 1 is the smallest, and the t_{diff} from Method 3 is the largest. The largest difference is less than 0.16 msec on all the three computer systems. This means that, by choosing different methods, we may introduce a systematic error of up to 0.16 msec. On the other hand, because the maximum difference is so small, all three methods are reliable enough for RT measurements.

Since we can rely on the accuracy of the device time $t_{receive}$ in Equation 1, if we performed clock speed correction, we know that the real t_{diff} is between $\max(t_{pre} - t_{receive})$ and $\min(t_{post} - t_{receive})$. So $\min(t_{post} - t_{receive}) - \max(t_{pre} - t_{receive})$ gives us the confidence interval of t_{diff}. In the middle rows of Figures 6F, 6G, and 6H, the difference of t_{diff} between Methods 3 and 1 tells us how small t_{diff} could be if we use Method 1 and how large it could be if we use Method 3.

RT Measurements Without External Triggers

To verify the accuracy of RT measurements without external triggers, we measured the "RT" of an ideal observer, the photodiode, without external triggers. The stimulus onset time t_{onset} (see Figure 4A) was returned by the Screen('Flip') function of PsychToolbox Version 3 on the basis of the computer clock. It is the predicted time at which the video display starts displaying the video frame that contains the stimulus. Figure 7 shows a histogram of the RTs of the ideal subject. We used Method 1 with 20 repeats for clock synchronization on all the three systems. The median RTs, the light time relative to t_{onset}, are 5.79, 5.64, and 5.67 msec for the three systems. Since the test stimulus was located in the center of the computer screen, at the 85-Hz refresh rate we used, the expected time difference between the light signal and t_{onset} is about half of a video refresh interval—that is, 1,000/85/2 = 5.88 msec.

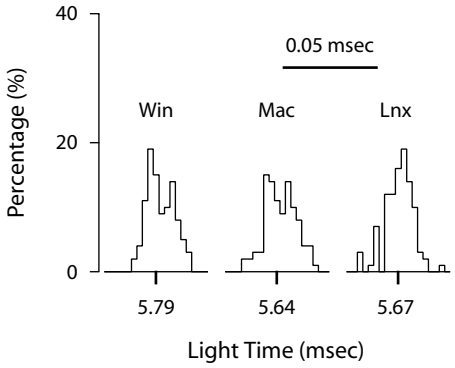

Figure 7. "Response time" measured without external trigger. The histogram shows the percentage of light signal time measured from the light port, relative to stimulus onset returned by the Screen('Flip') function of PsychToolbox 3. One hundred trials were collected on each computer system. The tick at each horizontal axis indicates the median of the samples.

This accounts for the nonzero time we got. The extreme small variability (less than 0.05 msec) among the 100 trials for each system indicates that the measurements are very precise. The nonzero "RT" (t_{delay}) can be used to correct human subject RTs:

$$t_{resp} = t_{button} - t_{onset} - t_{delay}. \quad (5)$$

If users care about the absolute RT, Equation 5 should be used instead of Equation 3. The procedure used here also serves as an example of how to do calibration using the light signal—that is, how to measure the time offset between the nominal stimulus onset time (t_{onset}) and the real onset time (light time). The principle is similar to that used by Black Box Toolkit (Plant, Hammond, & Turner, 2004), which can be used to verify timing accuracy and provide necessary corrections where practicable.

It is also interesting to compare the difference of "light RT" among the three systems. Although the differences are very small, with a maximum of 0.15 msec between the Win and Mac systems, all three "RTs" are significantly different ($p \approx 0$). Since we used the same CRT without changing photodiode location when we switched from one system to another, we would expect the "light RT" to be the same across three systems. There are several possible sources that may contribute to the small differences. The first is different refresh rates among the systems. The nominal rates are 85 Hz for all systems. However, the exact rate may be slightly different from 85 Hz. The second source may be the systematic time difference returned by Screen('Flip') on different systems. Whatever the reason is, this result demonstrates the power of the RTbox to detect small time differences. However, we cannot exclude the possibility of small systematic differences among the systems during clock synchronization.

DISCUSSION

Our tests indicate that the RTbox is a reliable tool for measuring RTs with very high accuracy. If a stimulus has a synchronized TTL trigger, the RT measurements are totally independent of the host computer. A 0.1-msec accuracy can be achieved. For visual stimuli, a built-in light port is used to receive trigger signals from a photodiode to guarantee the high accuracy. The TTL and light inputs can also be used for calibration purposes.

If external triggers are not available, our test results (Figure 6) indicate that we can get very good clock synchronization on all major computer systems. However, because the accuracy of clock synchronization depends on the particular host computer system and user program, it is necessary to verify the accuracy for a specific hardware and software setup. Our MATLAB driver gives a warning message if the synchronization is not accurate.

Although the methods we used to verify the accuracy of the RTbox are a little complicated, using the RTbox in experiments is actually very easy. If external triggers are available, clock synchronization is not necessary, and the accuracy of RT measurements is guaranteed. If no external trigger is available, clock synchronization is needed. The driver program can detect and report possible inaccuracies, so users can adjust the experimental setup to improve the timing accuracy.

Here, we discuss some limitations of the RTbox and possible solutions and improvements.

The microprocessor in the RTbox detects port and button events within a 0.1-msec interval. When an event is detected, it gets its time stamp and sends the data to the host computer. This process takes time and may slow down the detection of subsequent events. Theoretically, this is a problem because it may introduce timing errors. However, this is not a practical problem. One reason is that the time delay is very short (less than 1 msec to transmit the data of an event). Another reason is that, even if a trigger and a response or any two responses are so close in time, we do not normally treat them as valid responses. The refractory time could be a problem during calibration (such as those in Figure 5C), since we may try to detect two events that are very close in time.

The RTbox is designed with four buttons. Some experiments may need more buttons. The microprocessor chip is capable of detecting more than four buttons. Building an RTbox with more buttons requires changes to its hardware, firmware, and MATLAB driver code.

In some experiments, the user may want to measure RTs to auditory stimuli. Using the autodisable feature of the RTbox, we can easily deal with the fluctuations in sound waves and use them as triggers. We have already designed a new RTbox that will include a sound port to detect onsets of auditory stimuli.

The mechanical lag of buttons is another important factor in designing RT measurement devices. If a button has shorter RTs for presses, it normally has longer RTs for releases, and vice versa. This makes it hard to have buttons with short responding times for both buttonpresses and releases. In the next version, we will design a port to allow users to connect their specialized buttons, or button-driven TTL signals, to the RTbox for their special needs, such as an MRI-compatible keypad. We have also designed a

TTL output port to register all trigger and button events. This is very useful for EEG recording. The newly designed RTbox will also detect TR (repetition time) triggers from MRI scanners and pass them to the TTL output port. This is useful for simultaneous MRI and EEG recording. So in addition to its functionality in accurate RT measurements, the next version of the RTbox will also serve as an interface for EEG and simultaneous EEG/MRI recording systems.

AUTHOR NOTE

We thank Jie Zheng for valuable suggestions and for writing the first version of the firmware. This research was supported by NEI and NIMH. Correspondence concerning this article should be addressed to Z.-L. Lu, Department of Psychology, University of Southern California, Los Angeles, CA 90089-1061 (e-mail: zhonglin@usc.edu).

REFERENCES

Arieh, Y., & Marks, L. E. (2008). Cross-modal interaction between vision and hearing: A speed–accuracy analysis. *Perception & Psychophysics*, **70**, 412-421.

Brainard, D. H., Pelli, D. G., & Robson, T. (2002). Display characterization. In J. P. Hornak (Ed.), *Encyclopedia of imaging science and technology* (pp. 172-188). New York: Wiley.

Brown, S. D., Marley, A. A., Donkin, C., & Heathcote, A. (2008). An integrated model of choices and response times in absolute identification. *Psychological Review*, **115**, 396-425.

Chambers, C. D., & Brown, M. (2003). Timing accuracy under Microsoft Windows revealed through external chronometry. *Behavior Research Methods, Instruments, & Computers*, **35**, 96-108.

De Clercq, A., Crombez, G., Buysse, A., & Roeyers, H. (2003). A simple and sensitive method to measure timing accuracy. *Behavior Research Methods, Instruments, & Computers*, **35**, 109-115.

Dosher, B. A. (1976). The retrieval of sentences from memory: A speed–accuracy study. *Cognitive Psychology*, **8**, 291-310.

Jastrow, J. (1890). *The time relations of mental phenomena*. New York: Hodges.

Li, X., Lu, Z. L., Xu, P., Jin, J., & Zhou, Y. (2003). Generating high gray-level resolution monochrome displays with conventional computer graphics cards and color monitors. *Journal of Neuroscience Methods*, **130**, 9-18.

Lu, H., Morrison, R. G., Hummel, J. E., & Holyoak, K. J. (2006). Role of gamma-band synchronization in priming of form discrimination for multiobject displays. *Journal of Experimental Psychology: Human Perception & Performance*, **32**, 610-617.

Luce, R. D. (1991). *Response times: Their role in inferring elementary mental organization*. New York: Oxford University Press.

McKinney, C. J., MacCormac, E. R., & Welsh-Bohmer, K. A. (1999). Hardware and software for tachistoscopy: How to make accurate measurements on any PC utilizing the Microsoft Windows operating system. *Behavior Research Methods, Instruments, & Computers*, **31**, 129-136.

Plant, R. R., Hammond, N., & Turner, G. (2004). Self-validating presentation and response timing in cognitive paradigms: How and why? *Behavior Research Methods, Instruments, & Computers*, **36**, 291-303.

Plant, R. R., Hammond, N., & Whitehouse, T. (2003). How choice of mouse may affect response timing in psychological studies. *Behavior Research Methods, Instruments, & Computers*, **35**, 276-284.

Ratcliff, R., & Smith, P. L. (2004). A comparison of sequential sampling models for two-choice reaction time. *Psychological Review*, **111**, 333.

Shimizu, H. (2002). Measuring keyboard response delays by comparing keyboard and joystick inputs. *Behavior Research Methods, Instruments, & Computers*, **34**, 250-256.

Stewart, N. (2006). A PC parallel port button box provides millisecond response time accuracy under Linux. *Behavior Research Methods*, **38**, 170-173.

Yap, M. J., Balota, D. A., Cortese, M. J., & Watson, J. M. (2006). Single- versus dual-process models of lexical decision performance: Insights from response time distributional analysis. *Journal of Experimental Psychology: Human Perception & Performance*, **32**, 1324-1344.

APPENDIX

The latest information about the response box can be found at http://lobes.usc.edu/RTBox/. Users can download the MATLAB driver code, demo code, and firmware from the Web site.

MATLAB demo codes are RTBoxdemo.m for response time measurement without an external trigger and RTBoxdemo_lightTrigger.m for measurement using a light trigger.

```
function RTBoxdemo(scrn)

% This is a demo showing how to measure reaction time using RTBox.
% Run the program. When you see flash on screen, press a button.
% Your RT will be plotted after the assigned number of trials.
% Xiangrui Li, 3/2008

if nargin<1, scrn=max(Screen('screens')); end % find last screen
ntrials=10;  % # of trials
timeout=1;   % timeout for RT reading
sq=[0 0 100 100];   % square
rt=nan(ntrials,1);
RTBox('clear');  % in case it has not been initialized

try   % avoid dead screen in case of error
   [w rect]=Screen('OpenWindow',scrn,0);  % open a dark screen
   sq=CenterRect(sq,rect);
   HideCursor;
   ifi=Screen('GetFlipInterval',w); % flip interval

   % print some instruction
   Screen('TextSize',w,24); Screen('TextFont',w,'Times');
   str='This will test your response time to flash at the center of the screen.';
```

APPENDIX (Continued)

```
    DrawFormattedText(w,str,'center',rect(4)*0.4,[255 0 0]);
    str=sprintf('We will do %d trials. When you see a flash, press a button as soon as possible.',ntrials);
    DrawFormattedText(w,str,'center',rect(4)*0.45,[255 0 0]);
    DrawFormattedText(w,'Press any button to start', 'center', rect(4)*0.55, 255);
    Screen('Flip',w); % show instruction

    Priority(MaxPriority(w));   % raise priority for better timing
    RTBox(1000);  % wait 1000 s, or till any enabled event
    vbl=Screen('Flip',w);  %#ok turn off instruction

    for i=1:ntrials
        WaitSecs(1+rand*2); % random interval for subject
        Screen('FillRect',w,255,sq);
        RTBox('clear'); % clear buffer and sync clocks before stimulus onset
        vbl=Screen('Flip',w);  % show stim, return stim start time
        Screen('Flip',w,vbl+ifi*1.5); % turn off square after 2 frames

        % here you can prepare stim for next trial before you read RT
        t=RTBox(timeout);  % computer time of button response

        % check response
        if isempty(t), continue; end % no response, skip it
        t=t-vbl;  %  response time
        if length(t)>1 % more than 1 response
            fprintf(' trial %2g: RT=',i); fprintf('%8.4f',t); fprintf('\n');
            ind=find(t>0,1);  % take the first proper rt in case of more than 1 response
            if isempty(ind), continue; end  % no reasonable response, skip it
            t=t(ind);
        end
        rt(i)=t; % record the RT
    end
catch %#ok
    Screen('CloseAll'); Priority(0);
    rethrow(lasterror); %#ok
end
Screen('CloseAll');
Priority(0); % restore normal priority

rt=rt-ifi/2; % rough calibration: stim was at the center of screen

% plot result
h=figure(9); set(h,'color',[1 1 1]);
plot(rt,'+-');
set(gca,'box','off','tickdir','out');
ylabel('Response Time (s)'); xlabel('Trials');
rt(isnan(rt))=[]; % remove NaNs due to missed trials
str=sprintf('Your median RT: %.3f %s %.3f s',median(rt),char(177),std(rt));
title(str);

function RTBoxdemo_lightTrigger(scrn)
% This is a demo showing how to generate a flash as light trigger,
% and use RTBox to measure reaction time.
% In this demo, the stimulus is random noise at the center of screen,
% and light trigger is a 1-frame flash at right edge. You need to mount the
% light sensor to the flash position for this to work.

% Xiangrui Li, 3/2008

if nargin<1, scrn=max(Screen('screens')); end % find last screen
ntrials=10 ;  % # of trials
timeout=1;    % timeout for RT reading
trigsz=[40 80]; % trigger height and width
stimsz=120; % stim square size
efactor=3;  % larger noise checkers
```

APPENDIX (Continued)

```
stimdur=0.2; % stimulus duration
trigsq=ones(trigsz)*255;   % bright trigger square
csz=round(stimsz/efactor); % # of checkers
stim=uint8(rand(stimsz^2*2,1)*255); % random noise stimulus
rt=nan(ntrials,1);
RTBox('enable','light'); % enable light detection for trigger, also open device if needed

try   % avoid dead screen in case of error
   [w r]=Screen('OpenWindow',scrn,0);  % open a dark screen
   ifi=Screen('GetFlipInterval',w);% flip interval

   % print some instruction
   Screen('TextSize',w,24); Screen('TextFont',w,'Times');
   str='This will measure your RT to noise square at the center of the screen.';
   DrawFormattedText(w,str,'center',r(4)*0.4,[255 0 0]);
   str=sprintf('We will do %d trials. When you see the noise, press a button as soon as possible.',ntrials);
   DrawFormattedText(w,str,'center',r(4)*0.45,[255 0 0]);
   DrawFormattedText(w,'Press any button to start', 'center', r(4)*0.55, 255);
   Screen('Flip',w); % show instruction

   trig_tex=Screen('MakeTexture',w,trigsq); % make trigger texture
   % trigger position
   % trigrect=[r(3)-trigsz(2)-10 50 r(3)-10 trigsz(1)+50]; % top-right
   trigrect=[r(3)-trigsz(2)-10 r(4)/2-trigsz(1)/2 r(3)-10 r(4)/2+trigsz(1)/2]; % middle-right
   % trigrect=[r(3)-trigsz(2)-10 r(4)-trigsz(1)-50 r(3)-10 r(4)-50]; % bottom-right

   nframe=round(stimdur/ifi);  % # of frames of stim
   ClockRandSeed;   % set random seed
   for im=1:nframe % make stim textures for first trial
      stimsq=Expand(RandSample(stim,[csz csz]),efactor);
      tex(im)=Screen('MakeTexture',w,stimsq);
   end

   Priority(MaxPriority(w));   % raise priority for better timing
   RTBox(1000);  % wait 1000s, or till any enabled event
   Screen('Flip',w); % remove instruction

   for i=1:ntrials
      WaitSecs(1+rand*3); % wait for 1 to 4 s randomly
      RTBox('clear',0); % clear buffer before stim onset
      Screen('DrawTexture',w,tex(1));  % draw stim for 1st frame
      Screen('DrawTexture',w,trig_tex,[],trigrect); % draw trigger
      Screen('Flip',w); % show first frame: stim+trigger

      for im=2:nframe % draw noise for each frame
         Screen('DrawTexture',w,tex(im));  % draw frames
         Screen('Flip',w); % show stim, no trigger anymore
      end
      Screen('Flip',w); % turn off stim

      % prepare stim for next trial before reading RT
      Screen('Close',tex(:)); % release memory
      for im=1:nframe
         stimsq=RandSample(stim,[csz csz]);
         stimsq=Expand(stimsq,efactor);
         tex(im)=Screen('MakeTexture',w,stimsq);
      end

      t=RTBox('light',timeout);  % read RT, relative to light trigger
      if isempty(t), continue; end % no response, skip it
      if length(t)>1
         fprintf(' trial %2g: RT=',i); fprintf('%8.4f',t); fprintf('\n');
         ind=find(t>0,1);  % find the first proper rt in case of more than 1 response
         if isempty(ind), continue; end  % no reasonable response, skip it
```

APPENDIX (Continued)

```
            t=t(ind);
        end
        rt(i)=t;
    end
catch
    Screen('CloseAll'); Priority(0);
    rethrow(lasterror);
end
Screen('CloseAll');
Priority(0);                % restore normal priority

% plot result
h=figure(1); set(h,'color',[1 1 1]);
plot(rt,'+-');
set(gca,'ylim',[0 0.4],'box','off','tickdir','out');
ylabel('Response Time (s)'); xlabel('Trials');
rt(isnan(rt))=[]; % remove NaNs due to missed trials
str=sprintf('Your median RT: %.3f %s %.3f s',median(rt),char(177),std(rt));
title(str);
```

(Manuscript received April 20, 2009;
revision accepted for publication June 20, 2009.)