Using Arduino Microcontroller Boards to Measure Response Latencies

Thomas W. Schubert[a]

Alessandro D'Ausilio[b]

Rosario Canto[b]

[a]Instituto Universitário de Lisboa (ISCTE-IUL), Centro de Investigação e Intervenção Social, Lisboa, Portugal. Av. Forças Armadas, Ed. ISCTE-IUL, 1649-026 Lisboa, Portugal

[b]RBCS - Robotics, Brain and Cognitive Sciences Department, IIT - The Italian Institute of Technology

Accepted manuscript, now published as:
Schubert, T. W., D'Ausilio, A., & Canto, R. (2013).
Using Arduino microcontroller boards to measure response latencies.
Behavior Research Methods, 45, 1332–1346.
doi: 10.3758/s13428-013-0336-z

Email addresses: schubert@igroup.org, alessandro.dausilio@iit.it, rosario.canto@unife.it

Corresponding author is Thomas Schubert, schubert@igroup.org

Abstract

Latencies of button presses are a staple of cognitive science paradigms. Often keyboards are employed to collect button presses, but their imprecision and variability decreases test power and increases the risk of false positives. Response boxes and data acquisition cards are precise but expensive and inflexible alternatives. We propose to use open source Arduino microcontroller boards as an inexpensive and flexible alternative. These boards connect to standard experimental software using USB and a virtual serial port, or by emulating a keyboard. In our solution, an Arduino measures response latencies after being signaled the start of a trial, and communicates latency and response back to the PC over USB. We demonstrate the reliability, robustness and precision of this communication in six studies. Test measures confirm that the error added to the measurement has a *SD* of less than 1 ms. Alternatively, emulation of a keyboard results in similarly precise measurement. The Arduino performs as well as a serial response box, and better than a keyboard. In addition, our setup allows the flexible integration of other sensors and even actuators to extend the cognitive science toolbox.

Keywords: response latencies; reaction times; response box; Arduino; E-Prime

Using Arduino Microcontroller Boards to Measure Response Latencies

Paradigms that measure latencies of responses are a staple of many areas of experimental research on cognition. In theory, latencies of many different actions can be measured: key presses, spoken answers, or movements of fingers, hands, eyes, or other body parts.

Very often, however, experimental psychologists simply ask participants to react to computer-presented stimuli by pressing keys on regular keyboards. We believe this is unfortunate for two reasons: First, regular keyboards are notoriously imprecise input devices that add noise to the data. Second, key presses are most often arbitrarily matched to the outcome of some cognitive process, while other movements might index the investigated process more directly. Experimental psychologists may be aware of both shortcomings and still choose the keyboard out of habit, convenience, or lack of alternatives.

With the current work, we propose an alternative platform that allows both more precise measurement and more creative dependent variables. It uses the open source Arduino microcontroller platform and thereby harvests the creativity of a large developer community. In the following, we will elaborate on the imprecision of regular keyboards, before we describe and test how the Arduino can be used as a response latency measurement platform.

Keyboards and Response Boxes

Before we discuss the quality of response time measurement, we need to define the terms for doing so. Following Plant and Turner (2009), we will use the terms *accuracy* and *precision* to denote two different concepts. A response latency measurement is *accurate* when it is close to the actual time a response took. Imagine you measure how long it takes a 1 kg ball with a radius of 10 cm to roll down a 30°

slope for 1 m, repeating it many times (Galilei, 1638). If the average time is 903 ms, the measurement is *accurate*. Most likely your measures will be distributed normally around the average. The narrower the distribution, the more *precise* we call the measurement. It is possible to have a very precise measurement (a very low standard deviation) around an inaccurate average. This would be unfortunate for determining *g* from your measures, but it is typical for measures of response latencies in cognitive science when the focus is on differences between conditions, and less on the absolute size (within certain limits). Note however that there are situations where accuracy is more important, for instance when ratios instead of differences between latencies are investigated. We will focus on precision in this paper, but also reflect on accuracy when we have the numbers.

Even when using special software that synchronizes stimulus onset with the refresh cycle of the monitor, regular keyboards do not allow millisecond precise measuring of response latencies. Instead, the measured response latencies distribute normally around the true latency with a standard deviation that ranges from 1 to several milliseconds depending on the hardware (Voss, Leonhart, & Stahl, 2007).

For instance, Plant and Turner (2009)[1] found that a USB keyboard (Belkin) added an error with an average *M* = 18.30 ms and a standard deviation *SD* = 1.29 ms. Two other keyboards connected by serial port added different errors to the latencies, with *M* = 33.73, *SD* = 3.08, and *M* = 19.94, *SD* = 0.083. They also tested four computer mice connected by USB, and reported errors averages between 18 and 49 ms, and SDs between 0.7 and 4.28 ms. K. I. Forster and J. C. Forster (2003; see also J. C. Forster, 2012) tested one keyboard that produced measured latencies with a standard deviation of 5.15 ms; several USB mice ranged from *SD* = 5.65 to *SD* = 6.74, and two gamepads produced lower *SD*s (1.33 and 2.61). It is easy to imagine that with such a range of different distributions, using different equipment in a lab can quickly lead to erroneous results, and even sticking to one type of hardware adds noise to the data.

To overcome such imprecision, experimenters can employ special hardware that allows more precise measurement, for instance so-called response boxes. These response boxes provide a few keys in a special and fixed layout, and typically promise millisecond-precise measurement of latencies. Software for experimentation assures synchronization with the stimulus display (visual or audio) and records the response and its latency. Response boxes are connected to the computer either using a serial port, or via USB, depending on the model.

Increased precision of measurement provided by response boxes should translate into obvious methodological advances: Test power increases, and thus fewer participants and/or trials are needed to detect an effect, or smaller effects can be detected. At the same time, the risk of rejecting the null hypothesis because of a spurious result arising by chance should decrease.

Nevertheless, the use of response boxes is by no means universal. In many cases, this may be due to the robustness of typical paradigms: Some paradigms produce effects large enough to make the noise introduced by keyboards negligible. In other cases, however, researchers may be motivated to use response boxes, but do not because of the cost of the hardware (especially when running many participants in parallel) or technical difficulties in setting them up. Often, commercial response boxes only function (well) in concert with one particular software.

Viable alternatives to response boxes exist, but share some of the disadvantages. For instance, response boxes can be fashioned out of cheap equipment such as computer mice, and connected to the parallel port on a PC (Voss et al., 2007). However, parallel ports are disappearing fast from modern computers, and are not available on modern laptops, although alternative USB solutions have been proposed (Canto, Bufalari, & D’Ausilio, 2011). Also, multi-purpose hardware can be used to create response box functionality: Data Acquisition Cards are microprocessor boards that are plugged into expansion slots of desktop PCs, and allow digital input and output. They can be connected to external buttons and measure the time at which those buttons are pressed (Harmon-Jones, 2011). For instance, this is currently the only way to provide millisecond-precise response latency measurement in the free software package DMDX (K. I. Forster & J. C. Forster, 2003).

Empirical reports confirm the accuracy and precision of measurement that uses response boxes with serial connections, data acquisition cards, or parallel ports: The error added has typically a rather low average (around 6 ms), and a low standard deviation – in most cases less than or close to 1 ms (J. C. Forster, 2012; Plant, Hammond, & Whitehouse, 2003; Voss et al., 2007). Unfortunately, we know of no empirical report on the precision of USB-connected response boxes except the work by Li, Liang, Kleiner, and Lu (2010) on their own device. They showed a remarkable precision below 1 ms in concert with Matlab and the PsychToolbox software.

An obvious downside to response boxes is that they measure only presses (and releases) of buttons. While it would be unfair to caricature cognitive science as the study of beings with only one eye and two fingers, there is reason to believe that cognitive science overly relies on investigating choice decisions enacted with pressing one of two buttons. Unfortunately, measuring the latencies of moving fingers, hands, heads, or feet requires special hardware that costs even more, is even more difficult to set up than response boxes, and often only used by large labs that have access to a specialized technician.

To sum up: Response boxes provide much more precise measurement of the latency of button presses; yet, they are often not employed, and inherently restricted because of their specialized nature.

The Current Research

In this article, we propose to use the Arduino microprocessor platform as an alternative to keyboards and standard response boxes. We show how to connect an Arduino to standard reaction time software (using E-Prime), and report how precise the measurement is compared to standard response boxes. We also sketch how using the Arduino platform allows extending the experimental toolbox to include other measures beyond key presses. Thus, we describe how the Arduino can be used as a *platform* for measuring latencies of various kinds.

Our empirical strategy is to first evaluate how best to communicate between experimental software on a PC and an attached Arduino in Study 1, and then to develop scripts for E-Prime and Arduino that implement this communication. The robustness of these scripts is evaluated in Study 2. Study 3 tests the speed of our platform by having it detect onset of stimuli on a screen. Study 4 compares the speed of the platform to a serial response box and a standard keyboard by using a “robot” that detects stimuli onsets and hits buttons with a solenoid. Study 5 takes a closer look at increasing accuracy of latency measurement. Study 6 investigates the performance of a simpler version of communication that can be used with any experimental software, not just E-Prime.

The Arduino Platform

Arduino is the name of a family of microcontroller boards. The boards are a combination of an ATMEL microprocessor including RAM, flash memory, and input/output channels. Thus, these boards have the same general structure as common personal computers, but their performance is of course only a fraction of those. In the current paper, we mainly use the recent reference model of the Arduino family, the Arduino Uno. It has 32 kilobyte of flash memory, operates at 16 MHz, and features 14 digital input/output and 6 analog input pins. (Other, larger models provide more of those input and output options.) Part of the Arduino package is a programming environment, where code is written in a simplified C-like language, and transferred to the Arduino using a USB cable. After programming, an Arduino can work while being connected to a PC (and thereby transmit data), or operate standalone.

Several features make the Arduino family an interesting tool as a measurement platform. First of all, it connects easily by USB to a Windows PC, Mac, or Linux machine, and can transmit data using a virtual serial port to these operating systems. Second, it is open source hardware, which means that everybody can access, modify, and use the board design. Likewise, the software is free and open source. This has led to both a low price of a board (around 20-30 €, or 35$) and a large community that develops hardware and software compatible with the Arduino.

The boards can be connected to actuators and sensors, either commercial or build from scratch. Many extensions of a board come as so-called shields, additional small boards that are plugged into an Arduino board. For instance, some shields provide wireless access (WLAN or Bluetooth), while others allow storing data on flash memory cards. In addition, this idea of extending a board by plugging in components is taken further by rapid prototyping platforms, where smaller components are connected by plugging them into special connection shields. For instance, the prototyping platforms Tinkerkit and Seeedstudio Grove enable users to connect LEDs, small vibrating motors, buttons, accelerometers, gyroscopes, or hall sensors to the Arduino within minutes.

Together with the active community and freely available code, this guarantees a wide range of possibilities to create new kinds of measurements.

Despite its simplicity and low price, the Arduino can function as a reliable controller for experimental settings. When investigating how an Arduino can replace more complicated hardware as a standalone controller of experimental input and stimuli, D'Ausilio (2011) confirmed that signals generated by an Arduino were reliably constant in length and delay. He also found that typical combinations of input and output were performed with remarkable accuracy and precision, often with standard deviations of only microseconds. However, he did not yet explore how to embed Arduinos into response latency measures run by a PC.

Integrating Arduino into PC-Controlled Experimental Setups

With its speed and flexibility, Arduino boards could serve as useful platforms to build response devices that allow experimental setups with buttons, other sensors, and even output hardware. The challenge is to connect the Arduino to experimental software in a way that allows fast and reliable measurement.

In principle, there are several options to accomplish communication between an Arduino board and the computer that is executing the experiment. Here, we opt for a method that does not rely on additional or legacy hardware, namely to simply use the USB connection available on modern Arduino boards.[2] Most Arduino boards are outfitted with additional chips that convert the serial communication from the microprocessor into USB that connects then to a PC (these converter chips are Atmega16U2 on the Arduino Uno R3, Atmega8U2 on earlier models, or FTDI Serial-to-USB chips in older boards). Some Arduino boards come without such a connector, and it can then be added temporarily for programming. The more recent Arduino

Leonardo board uses the Atmel ATmega32u4 microprocessor that has built-in USB communication and thus does not require an additional converter chip.

Irrespective of the hardware, the result is that a USB cable connects Arduino and PC, and provides a serial connection. On the PC side, driver software creates a virtual serial (COM) port. This serial port can be accessed with any software that can communicate with a serial port. The driver offers several baud rates; for instance, the driver for the Uno offers 57600 and 115200 baud as the fastest speeds. Note that other boards will offer other baud rates, including 128000 baud.

Apart from choosing USB as the access hardware, there are at least two possible software strategies to access experimental hardware through the Arduino. One approach would be to use the Arduino as a relatively “dumb” gateway, another one would be to use its capabilities more fully, and in particular to have it measure latencies itself. Let us outline these approaches in more detail. As an example, imagine that two buttons are connected to the Arduino that are used as response keys in a reaction time study.

Treating the Arduino as a dumb gateway, its programming would simply continuously read the states of the two input pins to which the buttons are connected. When one of the buttons is pressed, this input is detected by the Arduino. It is programmed to then send a signal via the serial interface, which could simply consist of one character indicating which button was pressed. The connected PC, running the experiment, would present a stimulus and then listen on the serial port for the response to arrive (or until a timeout is reached).

Treating the Arduino as a microcomputer with timing capabilities, the procedure could take a different, second form. In this case, the PC would signal the Arduino via the Serial port when a response is expected (e.g., after presenting a stimulus to which the participant has to respond). When receiving this signal, the Arduino saves a timestamp, and starts waiting for the response. When the response is detected, the Arduino again takes a timestamp. It then transmits the difference as the response latency, and which response was given. This is the approach taken by the RTBox response box (Li et al., 2010).

The two approaches differ in several aspects. The first approach requires less programming. However, it is also less flexible; e.g., it would not afford the presentation of some stimuli or feedback via the Arduino itself. The two approaches also differ regarding robustness against lags in the serial connection. The first solution would only suffer from lag in serial communication from the Arduino to the PC, while the second solution would suffer only from lag in the serial communication from PC to Arduino. Study 1 was designed to investigate the lags of these two directions in order to decide which strategy is appropriate.

Study 1: Lags in USB Communication Between PC and Arduino

For this test, we connected an Arduino board via USB to a PC running E-Prime 2. Both the PC and the Arduino board were also connected to another PC with a data acquisition (DAQ) card equipped with a dedicated hardware clock. We had E-Prime send data to the Arduino, and the Arduino sent it then back. With the external clock in the DAQ card, we measured the duration of each leg.

Method

We used an Arduino UNO R1, and a PC (Intel Core 2 Quad Q6600, 2.4 Ghz) running Windows 7 and E-Prime 2.0.10.182. Serial port connections were set to 128000 baud. The external DAQ was a CED Micro 1401 mk II (from Cambridge Electronics Devices, UK) Sampling at 10000 Hz.

In each test trial, the sequence of events was as follows: (1) the E-Prime script sent an initial TTL pulse to the external timer to indicate the start of the trial. E-Prime then sent an integer (two bytes) over the Serial port to the connected Arduino. As soon as the Arduino received two bytes, it set one of its output lines to HIGH, thereby signaling the external clock connected to this line that it received data. It then waited for 300 ms (similar to a minimal response latency), and then sent the received integer back to the serial port. As soon as E-Prime received the integer, it again sent a TTL to the external clock, signaling the end of the roundtrip. We ran 990 of such trials.

Results

For the leg from E-Prime to Arduino, we measured durations that varied between 1.2 and 1.5 ms; *M* = 1.251 ms, *SD* = 0.051 ms. For the leg from Arduino back to E-Prime, durations varied between 1.9 ms and 4.9 ms, *M* = 3.080 ms, *SD* = 0.417 ms. In all cases, the received integer was the one that was sent.

These data suggest that sending data to the Arduino provides less absolute delay, but more importantly also a lot less variability of the delay. Thus, the more promising strategy is to (1) signal the Arduino the beginning of a trial, (2) have it measure the reaction time itself, and then (3) send the measured latencies back. This approach will be followed now, but we come back to the first approach in Study 6.

Connecting an Arduino Response Box to E-Prime

Scripting E-Prime to Interface Arduino

In the following, we will demonstrate how an Arduino can be integrated into a common software for running experiments, namely E-Prime. E-Prime has a built-in interface to serial ports. It does not, however, treat the serial port as a built-in input device. To integrate the Arduino, we thus use a script (called InLine in E-Prime) that communicates with the Arduino, and updates the response logs accordingly.

As a first step, a Serial device needs to be added to the Devices Tab of the experiment. We choose the Name Serial, 115200 Bits per second, 8 Data Bits, No Parity, and 1 Stop Bit as default values. The COM port and its specifications have to correspond to those assigned to the virtual serial port of the Arduino (which can be checked in Windows Device Manager, where also the baud rate can be adjusted to the one used in E-Prime).

Adding fields to the stimulus table.

We then set up stimulus presentations inside a list, as it is commonly done in E-Prime. Our list for the stimuli is called *StimulusList*, and for the stimulus presentation we use a TextDisplay Object named *trialStimulus*. In the *StimulusList* table, we add several additional properties: *TrialDuration*, which is transmitted to the Arduino and tells it the maximum time in milliseconds to wait for a response; this is the only additional field that the experimenter has to fill. Next, we add *TrialNr*, which will contain a random trial identifier number. We also add *ArduinoTrialNr*, *ArduinoRT*, *ArduinoButtons*, which are the three values that will be returned by the Arduino after a response, indicating the verification trial number, the response latency, and the pressed button(s), respectively. Finally, we add *InlineRT*, *InlineStartTime*, *InlinePostSendingTime*, and *InlineEndTime*, which are measured by our E-Prime script as a backup and control mechanism.

In addition, if feedback should be given, there should also be an attribute that lists the correct answer (we use the attribute *correct* for this purpose). Note that in our current setup, we only transmit integers to and from the Arduino, so the answer should be a positive integer between 0 and 32767 (or an integer between 0 and 9 if input will be tested with the keyboard as well). Figure 1 shows an example of this table.

Setup of the stimulus presentation object.

In the setup of the stimulus presentation object *trialStimulus*, the duration is set to a short value (we use 50 ms), and the PreRelease is set to the same value (entering *same as duration*). If feedback should be given, then Keyboard should be added as an

Input Device, and the list attribute containing the correct answer will be added in the Correct field (*[correct]*). Time Limit is set to *same as duration*.

Note that this arrangement implies that E-Prime closes the trialStimulus object after 50 ms, with two consequences. First, reaction time and response are set to null at this point, and have to be changed later. At the same time, these changes can only occur after that time. If reactions faster than 50 ms have to be measured, either the duration needs to be lower, or the script has to wait for a while.

Adding an InLine script after the stimulus presentation.

The *trialStimulus* is followed by an inline script that communicates with the Arduino. Setting the PreRelease to the same value as the duration assures that the inline script is executed immediately after the stimulus presentation is initiated. The script is shown in Listing 1 (posted as supplementary material)[3] and consists of four parts: (1) Setting up values to be transmitted, (2) Transmitting the values to the Arduino, (3) Waiting for and receiving the response from the Arduino, and (4) Disassembling the response and storing the response attributes.

As part of the setup, we fill *trialNr* with a random Integer value, giving each trial a unique identifier. In our example script, we send three integer values to the Arduino: (1) the trial number, which is also transmitted back by the Arduino as a control value; (2) the trial duration, and (3) a variable that could instruct the Arduino to do further action or output (and which we do not actually use in this example). The trial duration is read from the attribute TrialDuration set up earlier in StimulusList. Note that StimulusList is not referenced explicitly but through the Context object *c*. The three values are concatenated to one string, separated by commas, and ending with a dot. This string is written to the Serial device. Two timestamps are taken, before and after doing this (see Study 5).

The script then reads continuously from the serial device, adds any string that comes in to a temporary variable, and checks whether a dot is received – which is the stop signal from the Arduino. When that has happened, another time stamp is saved to document the process, but note that this is not the actual response latency. Instead, the script expects the Arduino to send back a list of three numbers: The trial number, the response latency measured by the Arduino in microseconds, and a number coding the answer registered in the Arduino.

The received string is then disassembled into these three separate numbers, which are stored in the corresponding StimulusList attributes (based on Margolis, 2011). Note that the response latency is of the type Long, which can be at maximum 2^{31}-1. As the unit is microseconds, the maximum response latency is thus 35 min. The two registered timestamps (also in microseconds) are also stored, along with the difference; this difference should obviously always be larger than the response latency registered by the Arduino.

Because the trialStimulus object is already finished by the time the script receives the response, at that moment it contains only a null response, and the log will contain empty fields for reaction time and response. This is a problem if feedback needs to be provided. Thus, if an answer was registered, the script corrects the saved null response by storing the response time (but now in milliseconds to comply with E-Prime standards) and the response itself also in the attributes .RT and .RESP of the stimulus object. It also compares the response to the correct response, and adjusts .ACC accordingly. Any following FeedbackDisplay can then use trialStimulus object as its input object.

Programming the Arduino to Interface E-Prime

Programs for the Arduino platform are called sketches. Our sketch (see Listing 2, supplementary material)[3] first sets up a serial connection (with the same values that are registered in Windows and E-Prime). It also sets up the pins to which the buttons are connected as input. It then waits for the transmission of three integers from E-Prime (separated by commas and ending with a dot). When these are received, it initiates a trial. At the beginning of a trial, the sketch stores a timestamp (*starttime*, in microseconds), and computes the maximum time by adding the read trial duration to the start time.

Using a *do… while* loop, the sketch then continuously checks for one of the inputs to go HIGH, or for the allotted time to end. When one of these events occurs, a time stamp is taken, and used to compute the response time (which is equal or larger to the allowed trial duration if no response was given).

The sketch then determines what should be given back as the response integer (0 for no response, 1 or 2 for one of the buttons, and 3 if both buttons were pressed simultaneously). Then, three integers are sent back: the trial number, the response time (in microseconds), and the response, again separated by commas and ending with a dot. This is the string that will be analyzed by E-Prime. Note that in the Arduino program, start time and end time are of the type unsigned long, which can be at most 2^{32}-1. The clock is started when the Arduino is reset by opening the serial connection at the beginning of the experiment; the timestamps will thus turn over after about 70 mins. Experiments that run longer than that should use alternative code that only measures in milliseconds.

Study 2: Checking the Robustness of the Communication

In the E-Prime and Arduino code that we presented here, there are two safeguards built in. First, a random number is transmitted for every trial to the Arduino, and given back by it; both are saved by E-Prime. Second, E-Prime not only stores the response latency that the Arduino measures, but also the time that passed between sending the trial to the Arduino, and receiving the answer back.

In our second test, we checked the robustness of these safeguards. Running the code described above, we repeated the test several times for different baud rates of the serial connection. Our goal was to test both whether the random number was always correctly transmitted, and how the standard deviation of the total duration was affected by serial port speed. In addition, we checked how long the Arduino waits in comparison to what it is instructed to wait.

Method

We used an Arduino Uno R3 for this test. An Atmega16U2 chip handles its USB connection. The board was connected to a Sony Laptop with Intel Core i5 processor, running E-Prime 2.0.10.242. We used the scripts introduced above. For each trial, the trial duration sent to the Arduino was 100 ms. Two buttons were connected but not pressed.[4] The Arduino thus only waited for the assigned 100 ms (while checking the button states), and then sent back the time it actually waited. We ran 500 consecutive trials for each of the following baud rates: 14000, 38400, 57600, 115200, and 128000 (all set in the Arduino code, E-Prime, and Windows' device manager).

Results

In all 2500 trials, the random number was correctly returned. The Arduino was instructed to wait 100 ms. To do this, it had to check the time it already waited in a loop. In the loop, it also continuously checked the state of both buttons. Out of all trials, it reported waiting 100 ms in 28.0% trials, 100.004 ms in 38.6% of trials, 100.008 ms in

32.4% of trials, and 100.012 or 100.016 in .9% of trials. (The resolution of micros() is 4 µs on 16 MHz Arduino boards.)

To compute the delay introduced by the USB communication, we subtracted the time the Arduino reported having waited from the response latency as measured by E-prime – the time between sending to and receiving from the serial port. Table 1 shows that this delay decreases steadily with increasing baud rate, from 22.73 ms with a baud rate of 14400 to 5.67 ms with 128000 baud. The standard deviation is also affected, dropping from 1.40 to 1.11.

Remember that the total delay is composed of the communication to the Arduino and the communication back from the Arduino. When using Arduino as a response platform, the communication to the Arduino will be performed in addition the response latency measured by the Arduino. These data show that in order to keep the added time as short and as least variable as possible, and in order to have these measures as a backup, the highest possible baud rate should be set. Furthermore, the data also assure that with a similar setup, the added noise is quite low even when factoring both legs of communication.

In sum, this test confirms the robustness of the code in terms of accurately communicating values back and forth. Note that a similar check can be performed with any data set collected with the code above: One can always subtract the Arduino-measured RT from the inline-measured RT, and check the average and variability of this delay. Finding low values for the roundtrip assures even lower values for its first leg.

Study 3: Testing the Variability of the Response With a Photo Diode

The previous test did not actually involve any response; the Arduino simply waited for the allotted time. In a next step, we intended to test speed and variability of the script and setup with an actual measurement by the Arduino.

Method

We connect a photodiode circuit to one of the pins of an Arduino Uno R2. The circuit compares the state of a photodiode (Osram BPW 34) to the resistance by a potentiometer using a comparator (Fairchild LM 311), and sets the connected pin HIGH or LOW accordingly.[5] The photodiode was placed on the upper half of a LCD flat screen (ASUS VE278), running at 60 Hz, connected by HDMI to a Sony Core i5 laptop. The potentiometer was adjusted such that the pin was HIGH when the screen was white, but LOW when the screen was dark blue. The Arduino sketch was changed such that it analyzed the state of the pin connected to the photodiode circuit. Either a HIGH value or a timeout resulted in the sketch returning response time and response.

On the E-Prime side, a script presented test stimuli with either a white or a dark blue background, which always followed a black screen that was presented for 300 ms. There were in total 500 trials with a white background and 500 trials with a dark blue background in random order. The Arduino received a waiting time of 250 ms. Duration of the trial presentation was set to 50 ms, and pre release to *same as duration*. Because the reactions from the Arduino were faster than those 50 ms, the script also waited for an additional 100 ms after receiving the response to allow the trial object to finish before changing .RT and .RESP in its logs. Baud rate was set to 128000 baud.

Results

In all of the 1000 trials, the photodiode correctly identified the screen as white or dark blue. One trial however showed an outlier low response time of 12 microseconds (for a white trial), probably due to volatility in the photo diode, and was removed from the analysis. The data for the blue trials resemble Study 2, and we do not report them here.

For the white trials, the response time reported by the Arduino ranged from 5.43 to 6.68 ms, *M* = 6.031 ms, *SD* = 0.117 ms. We again computed a delay as the difference between the response time measured by the e-prime script (including both legs of communication) and the Arduino-measured response time. The delay varied between 3.12 ms and 10.43 ms, *M* = 5.350, *SD* = 1.176 ms.

Assuming the screen had a perfect constant switch time from black to white, these data show that when using the full potential of the Arduino, it is possible to measure these switching times with a standard deviation of little more than a tenth of a millisecond. Even if the timing is left to E-Prime itself, we still only get noise with a *SD* of about 1 ms.

One might think that the *SD* = 0.117 ms overestimates the precision because the communication from E-Prime to the Arduino has an additional variability that is not captured here. However, note that to then explain the low variability of the times measured by the Arduino, one would have to assume that the true switching of the screen *counteracted* the variability in the communication, which is very unlikely. Thus, the tenth of a millisecond is likely close to the true precision.

Study 4: Comparing an Arduino Response Box to Other Response Boxes

Test 3 was instructive concerning the absolute performance of the Arduino platform; however, a comparison to commercial response boxes would be interesting. In Study 4 we thus compare the accuracy of a response box built using the Arduino to a standard serial response box and a keyboard, always using E-Prime.

Method

To improvise an Arduino response box, we connected two Cherry MX key modules to the Arduino in the standard way (http://arduino.cc/en/Tutorial/Button), using an Arduino-compatible rapid prototyping kit (Seeed Studio Grove Base Shield and its screw terminal connector). The Arduino sketch was adapted such that it continuously read the states of both input pins and the timeout. The key modules were mounted on Lego Duplo bricks for a solid foundation (see Figure 3).

As a comparison, we used a PST Serial Response Box connected via a serial port to a desktop PC running Windows 7 and E-Prime 2.0. We also compared the performance of a standard USB Microsoft keyboard.

We created an E-Prime experiment that displayed 100 very simple trials: In each trial, a black waiting screen was followed by a white target screen. The program then waited for a button on the response device to be pressed. To press the button in response to the white screen onset, we constructed a “robot”, using a *second* Arduino that was not connected to the computer itself. This Arduino featured the photodiode circuit also used in Study 3, placed on the screen (Asus VG236H set to 60 Hz). In addition, it was connected to a 12 V push solenoid (Intertec ITS-lz-2560 d-12vdc). This solenoid was placed above the button of the response box, pushing it when turned on. The sketch running on this second Arduino continuously checked whether the photodiode pin was set HIGH or LOW; when it was HIGH, the solenoid was fired, which pressed the button and triggered the end of the trial, releasing the solenoid. The code reading the photodiode was adapted such that a change in brightness was only registered if two continuous readings resulted in the same value, to make the program more robust.[5]

Results

We ran two tests of 100 items with each hardware; we report both separately to convey the reliability of the measurement. Because we compared the Arduino measurement to the E-Prime measurement for the two devices, the Arduino also only measured in milliseconds.

The crucial numbers in this test are the standard deviations. The total reaction times should be interpreted with caution because they are based on completely different hardware, and it is possible that the solenoid hit the different buttons in slightly different ways, with different travel distances, etc.

The standard in this test is set by the Serial Response Box, usually regarded as millisecond-precise. Table 2 shows indeed two standard deviations between 1 and 1.4 ms; the range of responses is impressively low. These small numbers confirms that our Arduino-solenoid-robot is also performing quite reliably – at least with the precision found here. The overall latency is rather high, between 56 and 58 milliseconds. We assume that this is due to the large travel of the solenoid and the specifics of the circuit.

Our Arduino response platform performs almost as well as the PST Serial Response Box, with two *SD*s of 1.28 and 1.29 ms. This shows that the combination of PC-to-Arduino USB communication and having the Arduino measuring the RT makes up for the disadvantage of having no real direct serial connection to the PC.

The keyboard performs worse than the two other devices, but still with an acceptable noise *SD* of about 3 ms. Note however that that this cannot be generalized to USB keyboards in general, and hardware details can vary even within one product line.

To compare the variance in the various tests, we computed Levene’s tests for equality of variances. Regarding their variances, the two Arduino measures did not differ from each other, and neither did the two keyboard measures, both Fs < 1. However, the variances of the two response box measures did differ, $F(1,198) = 4.63$, $p = .033$.[6] Importantly, the two Arduino measure variances did not differ from the two response box variances, Fs(1,198) < 1.81, ps > .180. The keyboard variances differed from all other variances, Fs(1,198) > 51, ps < .001. Figure 4 shows the distributions. In sum, both Serial Response Box and Arduino performed much better than a keyboard;

and the difference between Arduino and Response Box seem to be not larger than the difference between two different tests with the Response box.

Study 5: Estimating the Delay from Trial Initiation in E-Prime to Start of Clock in Arduino

In the script developed here, we send a signal from E-Prime to the Arduino at the start of a trial. This signal prompts the Arduino to start its time measurement. Obviously, the time the signal takes to travel from E-Prime to the Arduino is not included in the final response latency reported back by the Arduino. In Study 1 we found that the average of this additional delay was 1.25 ms, with a very low *SD*. Nevertheless, ignoring this delay reduces the accuracy of the measurement, and may be undesirable in some contexts.

Above, we mentioned that the E-Prime scripts stores two timestamps, one immediately before sending the signal, and another one upon receiving the data back from the Arduino. By testing how much longer this difference is than the latency reported by the Arduino, researchers can judge the reliability of the USB speed, but this delay will include both legs of communication, not the first delay in particular.

This problem was already tackled by Li et al. (2010) in the development of their RTbox. Their software similarly sends a start trial signal to an external microprocessor, and records timestamps right before and after that operation. In addition, their microprocessor software takes and reports a timestamp upon receiving the trial start signal. This differs slightly from our approach so far, because on the E-Prime side, we only analyzed the timestamp before sending a signal, and on the microprocessor side, we only report back the time difference.

Li et al. discussed three different methods for estimating the true moment of sending the signal from computer to microprocessor. Each relies on taking several samples (at least 20). One possibility is to compute the difference of the two computer timestamps before and after sending, and to select the shortest one. This has to include the true time of sending the signal, and to estimate that moment, Li et al. simply average the two timestamps associated with that shortest delay.

In another method, they compute the difference between the computer timestamp before sending and the microprocessor timestamp upon receiving. Note that these will have an unknown offset because the clocks of the two processors are started at different times. Li et al. then select the minimum difference of the two timestamps, and then use the associated computer timestamp before sending as an estimate of the true moment of sending.[7]

In order to demonstrate how the methods devised by Li et al. can be applied in our scripts as well, we ran another study in which we record both timestamps before and after sending the serial signal, and investigate the differences between these timestamps and their distributions.

Method

On the Arduino side, we changed the script such that the timestamp of starting the trial is also returned, in addition to the latency. On the E-Prime side, we store this additional timestamp from the Arduino. We used an Arduino Leonardo, connected with 128.000 baud to E-Prime 2.0.10.242 running on a Windows 7 laptop.

We ran consecutive 300 trials. In each, E-Prime initiated a trial with 100 ms duration, saving timestamps before and after sending serial communication. The Arduino only waited the allotted time without any further measurement. In E-Prime, before each trial we added a random delay between 1 and 100 ms to desynchronize the process from the computer's USB cycles, as it would be the case in a normal experiment. Additional constant delays were present as well. The complete test took about 12 minutes.

Results

We first checked the delay between the timestamps taken by E-Prime before and after sending the serial communication. The average delay was 0.341 ms, ranging from 0.16 to 0.55 (*SD* = 0.087). Following Li et al., we selected the minimum delay and used the average of the two associated E-Prime-recorded timestamps (before and after) to estimate the true sending time. In our experiment, this arrived at an offset of 777411.246 µs between the two clocks (due to unequal starting times of the experiment on computer vs. Arduino).

We then investigated the difference between the E-Prime timestamp before sending, and the Arduino timestamp upon receiving the start trial signal. The average of this delay is uninformative because they are taken with two different unsynchronized clocks. The *SD* was rather large with 7.926 ms. However, note that these 300 measurements were taken over the course of 12 minutes. Li et al. (2010) have shown that due to small differences in the actual speed of clocks, drifts between them occur that have to be removed. When we regressed the differences on the number of trial, we found a large linear effect of trial number, $\beta = .999$, $t(299) = 495$, $p < .001$. After removing this linear influence, the unstandardized residuals had a much lower *SD* of 0.28 ms. Again following Li et al., when using the sending time associated with the shortest difference residual, we arrive at an offset of 777411.325 µs, almost identical to the first method.

Discussion

In the scripts above we ignored the communication delay between sending the trial start signal from E-Prime and receiving it in the Arduino. However, if concern arises or the paradigm requires it, the true time of sending, and thus the true response latency including the communication delay, can be *estimated* following the methods devised by Li et al. (2010). For this purpose, the two timestamps taken by E-Prime before and after sending can be used effectively. Essentially, this approach synchronizes the clock in the Arduino and the clock in the computer, which are always started at different times with an unknown offset.

Li et al. actually propose the synchronization of clocks in this manner before each trial, implementing between 20 and 100 communication cycles between computer and microcontroller in order to estimate the true RT. Note that while this increases *accuracy* of the latency measurement in terms of getting closer to the true value, it does not increase *precision*, because the measurement itself is again a new random sample from the distribution identified for the duration of communication.

Given that all our results (and theirs) show that this delay is shorter than 1 ms, we do not believe that this is necessary for most purposes. However, we definitely recommend checking both a) the duration of sending the serial communication and b) the difference between Arduino-reported latency and total E-Prime measured latency *for all trials* to assure against general malfunctioning and sudden drops in performance.

As already noted, this approach only estimates the delay between start of a trial and start of time measurement in the Arduino. To get an *accurate measurement* instead of an estimate, it would be necessary to program the Arduino such that it detects the onset of a trial itself. This could be implemented by signaling the onset of a trial with a cue on the screen (e.g., a dot in a corner), and equip the Arduino with a light sensor that allows it to save a timestamp at the true onset. While we will not implement this in the present paper, it can be easily done. Together with Study 3, the following study will provide a background for the precision of such an approach.

Study 6: Emulating Human Interface Devices with the Arduino Leonardo

In Study 1 we determined that communication from the computer to the Arduino board was faster and less variable than communication from the Arduino back to the computer. We therefore developed scripts that used only the first direction under time critical conditions, and then let the Arduino measure the response latency, and report it back when timing was not critical anymore. This allows the Arduino also to receive instructions for stimulus presentation. The cost is that we need to add special scripts to the latency-collecting software.

In our final Study 6 we also want to explore how an Arduino response box can be constructed without relying on specialized scripts. For this purpose we take advantage of the recently introduced Arduino Leonardo. This board does not feature a separate chip that handles the communication with USB. Instead, the main processor, an Atmel ATmega32u4, also handles this communication. In all other respects this board is similar to the Arduino Uno R3.

To estimate the performance of the Leonardo, we first repeated the fastest condition of Study 2 (128000 baud) with the Leonardo board. We encountered a higher percentage of longer waiting times, but note this concerned only microsecond differences: The program was scheduled to wait only 100 ms; it reported waiting exactly that long in 25% of trials. About half of the trials (47.2%) it waited 100.004 ms, in a fourth of the trials (26.4%) it waited 100.012 ms, and in 1.4% it waited 100.016 or 100.020 ms. The last line of Table 1 shows the delay, computed as the difference between time of the total roundtrip measured by E-Prime and the time waited by the Arduino. This delay has an average of 3.10 ms, with an *SD* of 0.53 ms, ranging from 1.97 to 9.22 ms. These values indicate that the distribution is quite narrow with only a few outliers. Indeed, 99.4% of the delays were below 3.72 ms, and there were three outlier values between 7.2 and 9.23 milliseconds. This suggests that the Leonardo could be a viable alternative to the Uno despite the fact that the main processor handles also USB communication. At least with simple tasks like in this test (checking the status of two input pins), USB communication seemed to be even faster than with an Uno, although there were 0.6% outliers with higher delays.

However, the main advantage of the Leonardo is another feature. Running USB connection in the main processor allows the Arduino to emulate a keyboard or a mouse when connected by USB. In other words, the Leonardo can be programmed to simulate the press of a keyboard button in response to events, and response latency software recognizes this button press as if it came from a regular keyboard. Of course, this signal still has to go through the regular polling process of the operating system, but there is a chance that noise coming from the construction of a regular keyboard is avoided in this way (Li et al., 2010).

Method

To evaluate this option, we connected an Arduino Leonardo to a light dependent resistor (LDR), in the form of the TinkerKit LDR Sensor sold by Arduino.cc. We intentionally chose this module to demonstrate accuracy using standard commercial components, and to our best knowledge there is no photo diode module for any of the Arduino rapid prototyping platforms. The LDR was placed in the right top corner of the screen of a Sony Vaio laptop. We programmed the Arduino such that it would continuously read the state of the LDR, and simulate the pressing of key A when the measurement surpassed the level 200.

Note that the LDR was connected to an analog pin on the Arduino. Reading such a pin takes longer than reading a digital pin, about 0.1 ms. We added a further delay of 10 microseconds to allow complete settling of the analog to digital converter. As a result, all internal reading on the Arduino should have values multiple of approximately 0.11 ms.

On the Windows side, we programmed a simple experiment in both E-Prime 2.0.10.242 and DMDX 4. In both cases, there were 200 trials. Each trial first showed a black screen for 2000 ms, and then a white screen, waiting for the pressing of the button A. The key press was measured with the regular functions in E-Prime and DMDX.

As a comparison, we added one condition in which we used the script developed above (E-Prime only because DMDX does not support a serial interface), thereby measuring response latency on the Arduino directly. In this case, the Leonardo was connected with a speed of 128.000 baud, and the script checked the state of the LDR.

Results and Discussion

Note that E-Prime usually reports measurements in milliseconds, while our E-Prime script and DMDX report in microseconds. When the Leonardo was programmed to return the letter A upon the screen turning white, DMDX recorded latencies between 5.53 and 6.69 ms, *M* = 6.09, *SD* = 0.31 (condition DMDX/Emulation). E-Prime reported latencies between 6 and 9 ms, *M* = 7.31, *SD* = 0.52 (condition E-Prime/Emulation). When E-Prime and Leonardo were programmed with the scripts developed in the current paper (Study 2), latencies ranged from 4.10 to 5.26 ms, *M* = 4.65, *SD* = 0.29 (condition Arduino Measurement). We compared the variances and means with Levene's tests and *t*-tests, respectively. DMDX/Emulation and Arduino Measurement did not differ from each other regarding variances, $F < 1$, but regarding means, $t(398) = 47.78$, $p < .001$. E-Prime/Emulation differed from both other conditions regarding variances, *F*s(398) > 94, *p*s < .001 and means, *t*s > 28, *p*s < .001.

These tests show a remarkable accuracy of the Arduino Leonardo even when it is used simply to emulate a keyboard device. We achieved *SD*s of half a millisecond in both E-Prime and DMDX. DMDX even rivaled the script developed above, as the *SD* of its measurements was not significantly higher than the measurements run on the Arduino itself. E-Prime performs less well, perhaps simply because it measures only in milliseconds, but still very good. This suggests that if the Arduino is not used to present other stimuli triggered by the communication it receives from E-Prime, using an Arduino Leonardo and emulating a HID device might be sufficiently accurate.

In addition, together with Study 3, these results suggest that adding a light sensor can be a valid approach to allow the Arduino registering the onset of a trial, and thereby increase accuracy of measurement.

General Discussion

Current response time measurement in cognitive science relies heavily on measuring latencies of key presses on standard keyboards. Although our Study 4 showed that the precision of such measurements could be acceptable, this hinges immensely on the particular keyboard and paradigm. Furthermore, key presses might not always be the ideal movement to measure. More precise response boxes are expensive and often rely on legacy interfaces or on specific software. Instruments that measure other movements are even more expensive and difficult to program.

In the present article, we described a way out of this dilemma. We show how the open source microcontroller platform Arduino can be combined with a standard software package, E-Prime, to measure response latencies as precisely as a serial response box does. We show that flexible millisecond precision is within the reach with hardware investments of less than 50 €.

In Study 1, we saw that communication in the direction from the PC to the Arduino is faster and less noisy than the return way. We therefore program the Arduino such that it gets signaled when a trial starts, measures the response latency itself, and then reports back both the latency and the response. We developed an E-Prime inline script that signals the trial to the Arduino, waits for the measured values, and saves them to the database. Study 2 showed that this communication is reliable and fast when high baud rates are selected. Study 3 confirmed that with this script, the setup of PC, Arduino, and E-Prime can measure an event (here a change on the screen) with excellent precision (an *SD* of < 1 ms). Study 4 confirmed that a response box built with this setup performs as well as a serial response box, the current gold standard, and better than a keyboard. Study 5 showed that by taking additional timestamps into account, accuracy can be enhanced. Finally, Study 6 demonstrated that even without using special scripts, the Arduino Leonardo board could be used to construct a millisecond precise response box that can be used with any software.

Our script includes two quality assurance mechanisms: a check of the correct match of E-Prime and Arduino trial via a random number, and a backup measurement of the latency via the script. Both should be used to verify the correct implementation of the script (see a check list in the Appendix, posted as supplementary material with the article).

We focused here on replicating response box and keyboard functionality because this is the current norm. However, the Arduino has potential for building new kinds of response measures. It can be connected to a multitude of sensors to develop new paradigms in an easy fashion. Rapid prototyping toolkits offer gyroscopes, accelerometers, light-dependent resistors, hall sensors, compasses, touch sensors, and much more. Here we only demonstrate the precision using keys (Study 4), photodiode (Study 3), and light-dependent resistors (Study 6). Tests with other hardware have to be conducted in the future.

Interestingly, the script that we developed here can be directly used to also implement the presentation of stimuli via the Arduino. For instance, in some paradigms vibration or LEDs have to be used (e.g., Pavani & Castiello, 2004). LEDs and small vibrators can easily be connected to an Arduino, and the Arduino can control them in response to the trial signal. We believe that controlling external actuator and measurement hardware from experimental software like E-Prime via an Arduino can greatly enhance the experimental psychologist's toolbox.

Caveats and Future Directions

Perhaps the biggest problem for anybody who wants to implement our solution right away is that in rapid prototyping kits for the Arduino, there seems to be shortage of good button modules. While this situation remains, we suggest three solutions. Most easily, any button can be connected to an Arduino directly.[8] We recommend employing Cherry key modules.

Second, it is clear that our E-Prime InLine script requires basic knowledge in scripting, and is not as easily deployed as simply choosing an input device in E-Prime. However, there are very few changes that experimenters need to make in a standard experiment. We are currently working on packaging the script such that it is more easily employed.

Third, we are certainly aware that other software packages exist that also implement serial ports (e.g., Presentation). We expect that our script can be easily ported to these packages. We make our script available under the GNU GPL v3 public license and encourage porting to other environments.

We focused here on standard Arduino USB communication, but we would like to point out that alternatives are available. For instance, it is conceivable to use parallel ports where they are still available to communicate with an Arduino, or to employ a

DAQ card. Another option is to circumvent the hardware that an Arduino offers for USB connection, because this is not optimized for extreme speeds. For instance, it is possible to connect the hardware serial port of an Arduino board to a special FTDI chip, which then talks to the computer via USB and specialized drivers. In initial tests we have found an improved accuracy using this setup.

Finally, we want to reiterate that despite its many advantages, the use of additional hardware may not be necessary in some paradigms. Studies on response latencies have been successfully conducted over the Internet, which presents the most noisy environments imaginable, with many different hardware configurations (Nosek, Banaji, & Greenwald, 2002). The challenge is to judge *a priori* based on expected effect size, desired test power, and response latency distribution, what noise induced by the particular hardware is acceptable. Providing guidelines for this remains a task for the future.

We hope that experimental psychologists increasingly abandon the keyboard as a response device when it is not the ideal index of cognition. For many researchers in cognitive science, programming their own paradigms has become an elementary part of developing studies. We believe that building suitable hardware can become similarly useful and natural.

References

Canto, R., Bufalari, I., & D'Ausilio, A. (2011). A convenient and accurate parallel Input/Output USB device for E-Prime. *Behavior Research Methods*, *43*(1), 292–6. doi:10.3758/s13428-010-0022-3

D'Ausilio, A. (2011). Arduino: A low-cost multipurpose lab equipment. *Behavior Research Methods*. doi:10.3758/s13428-011-0163-z

Forster, J. C. (2012). *DMDX Help Input*. Retrieved August 16, 2012, from http://www.u.arizona.edu/~jforster/dmdx/help/dmdxhinput.htm

Forster, K. I., & Forster, J. C. (2003). DMDX: a Windows display program with millisecond accuracy. *Behavior Research Methods, Instruments, & Computers*, *35*(1), 116–124.

Galilei, G. (1638). *Discorsi e dimostrazioni matematiche, intorno à due nuove scienze*. Leiden: Lodewijk Elzevir.

Harmon-Jones, E. (2011). *How to make a reaction time box for DMDX*. Retrieved November 20, 2011, from http://www.socialemotiveneuroscience.org/making_RT_box_DMDX.pdf

Li, X., Liang, Z., Kleiner, M., & Lu, Z.-L. (2010). RTbox: a device for highly accurate response time measurements. *Behavior Research Methods*, *42*(1), 212–25. doi:10.3758/BRM.42.1.212

Margolis, M. (2011). *Arduino cookbook*. Sebastopol, CA: O'Reilly Media.

Nosek, B. a., Banaji, M., & Greenwald, A. G. (2002). Harvesting implicit group attitudes and beliefs from a demonstration web site. *Group Dynamics: Theory, Research, and Practice*, *6*(1), 101–115. doi:10.1037//1089-2699.6.1.101

Pavani, F., & Castiello, U. (2004). Binding personal and extrapersonal space through body shadows. *Nature Neuroscience*, *7*(1), 14–16. doi:10.1038/nn1167

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*(2), 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*(2), 276–84.

Plant, R. R., & Turner, G. (2009). Millisecond precision psychological research in a world of commodity computers: new hardware, new problems? *Behavior Research Methods*, *41*(3), 598–614. doi:10.3758/BRM.41.3.598

Voss, A., Leonhart, R., & Stahl, C. (2007). How to make your own response boxes: A step-by-step guide for the construction of reliable and inexpensive parallel-port response pads from computer mice. *Behavior Research Methods, Instruments, & Computers*, *39*(4), 797–801.

Footnotes

[1] Using Windows XP SP2 and E-Prime 1.3, and the Black Box ToolKit (www.blackboxtoolkit.com), see Plant, Hammond, and Turner (2004).

[2] We can think of three alternatives: First, Arduino boards can be outfitted with serial connectors, which would allow connecting them to a PC with a serial input. Second, Arduino output pins can be connected to a data acquisition card's input pins or to a parallel port. Both would result in fast and flexible solutions, but also require legacy or additional hardware that we want to replace. A third solution would establish a wireless connection via WLAN or Bluetooth; however, time lags added to the transmissions would have to be evaluated carefully.

[3] All scripts and an E-Prime template file are also available at http://openscienceframework.org/project/cuzj5/

[4] This means essentially that a pull-down resistor connected the pin to ground. Reading an unconnected pin would result in random fluctuation.

[5] See https://reactiontimes.wordpress.com/electro-mechanical-turk/ for the layout and further details.

[6] The difference between the two tests for the Response Box is surprising but remains unclear. We ran two more tests with 100 trials each, but the results resemble those of the run with the larger variance.

[7] A third method, which is shown to be slightly inferior, uses the difference between microprocessor timestamp and computer timestamp after sending, but we will not discuss this here further.

[8] See http://www.arduino.cc/en/Tutorial/button for a tutorial.

Author Notes

The main part of this research was conducted at *Instituto Universitário de Lisboa (ISCTE-IUL)*, *Centro de Investigação e Intervenção Social*, Avenida das Forças Armadas, 1649-026 Lisboa, Portugal, with support from *Fundação para a Ciência e a Tecnologia* (FCT) grant PTDC/PSI-PSO/101366/2008. Thomas Schubert is now at University of Oslo. We thank Volker Seibt, Robert Richards, Gonçalo Paiva, Pedro Antunes, and João Fonseca for their help with hardware, and Carla Murteira, Elizabeth Collins, and Diniz Lopes for their help and comments on this project. Please address correspondence to Thomas Schubert, Psykologisk institutt, P.O box 1094 Blindern, 0317 Oslo, Norway, or schubert@igroup.org.

Tables

Table 1

Total delay introduced in USB communication (E-Prime measured response latency minus Arduino-reported waiting time; all values in milliseconds; Studies 2 and 6).

Arduino	Baud rate	*M*	*SD*	Minimum	Maximum
Uno R3	14400	22.7323	1.40497	18.48	26.21
	38499	10.6446	1.10654	7.23	12.71
	57600	8.1994	1.07971	5.56	10.76
	115200	5.7482	1.12930	3.28	8.37
	128000	5.6715	1.10581	3.40	7.82
Leonardo	128000	3.0997	0.52883	1.97	9.22

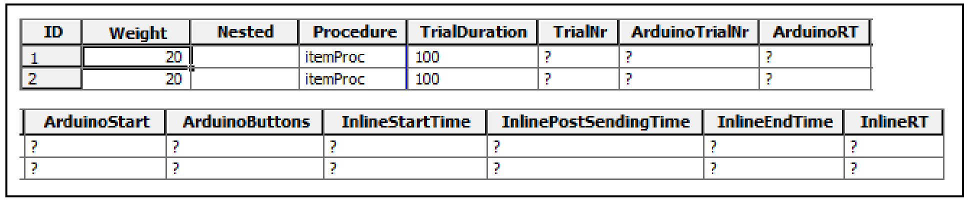
Table 2

Latencies of responses (in milliseconds) by the "robot" (Second Arduino equipped with photodiode and Solenoid) to react to onset of a white screen (Study 4).

Hardware	Test	*SD*	*M*	Min	Max
PST Serial Response Box	1	1.08	56.28	54	59
	2	1.34	57.54	54	61
Arduino	1	1.29	42.38	38	45
	2	1.28	41.87	40	46
Microsoft USB keyboard	1	2.99	69.67	63	77
	2	3.09	72.25	65	79

Figure 1

E-Prime Table (separated in two parts) with added fields for communication with and control of the Arduino: *TrialDuration*, *TrialNr*, *ArduinoTrialNr*, *ArduinoRT*, *ArduinoButtons*, *InlineStartTime*, *InlinePostSendingTime*, *InlineEndTime*, and *InlineRT*. Only *TrialDuration* has to be filled by the programming experimenter.

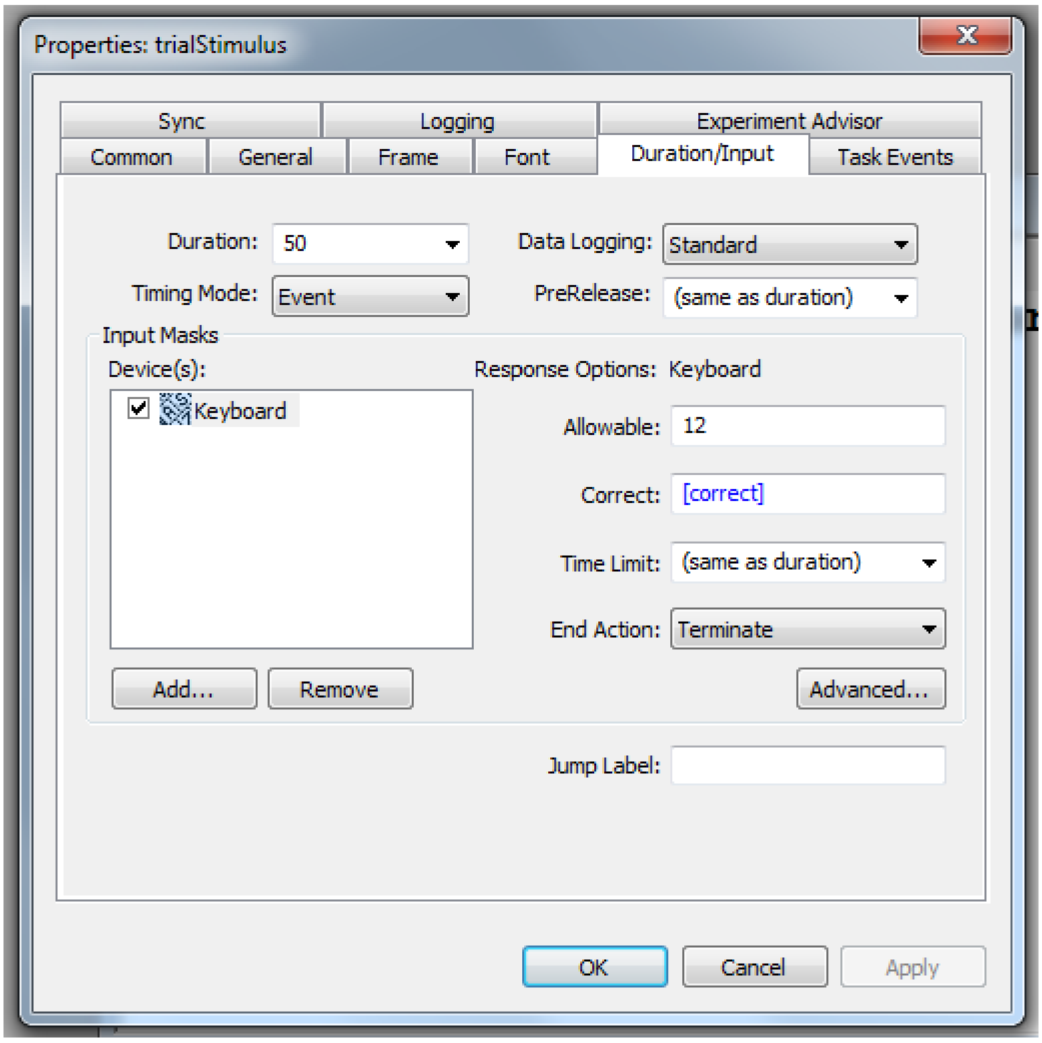


ID	Weight	Nested	Procedure	TrialDuration	TrialNr	ArduinoTrialNr	ArduinoRT
1	20		itemProc	100	?	?	?
2	20		itemProc	100	?	?	?

ArduinoStart	ArduinoButtons	InlineStartTime	InlinePostSendingTime	InlineEndTime	InlineRT
?	?	?	?	?	?
?	?	?	?	?	?

Figure 2

Properties of the E-Prime stimulus presentation object, here a TextDisplay. Duration is set to a very short value (actual duration is controlled by TrialDuration in the StimulusList table). *Keyboard* and *Correct* are added to allow feedback and testing. Data Logging is turned on.



Duration: 50
Data Logging: Standard
Timing Mode: Event
PreRelease: (same as duration)
Device(s): ✓ Keyboard
Response Options: Keyboard
Allowable: 12
Correct: [correct]
Time Limit: (same as duration)
End Action: Terminate
Jump Label:

Figure 3

Connecting two buttons to the pins 6 and 7 of an Arduino, realized with Cherry key modules (inset a), screw terminals (inset b), and a rapid prototyping base shield (inset c).

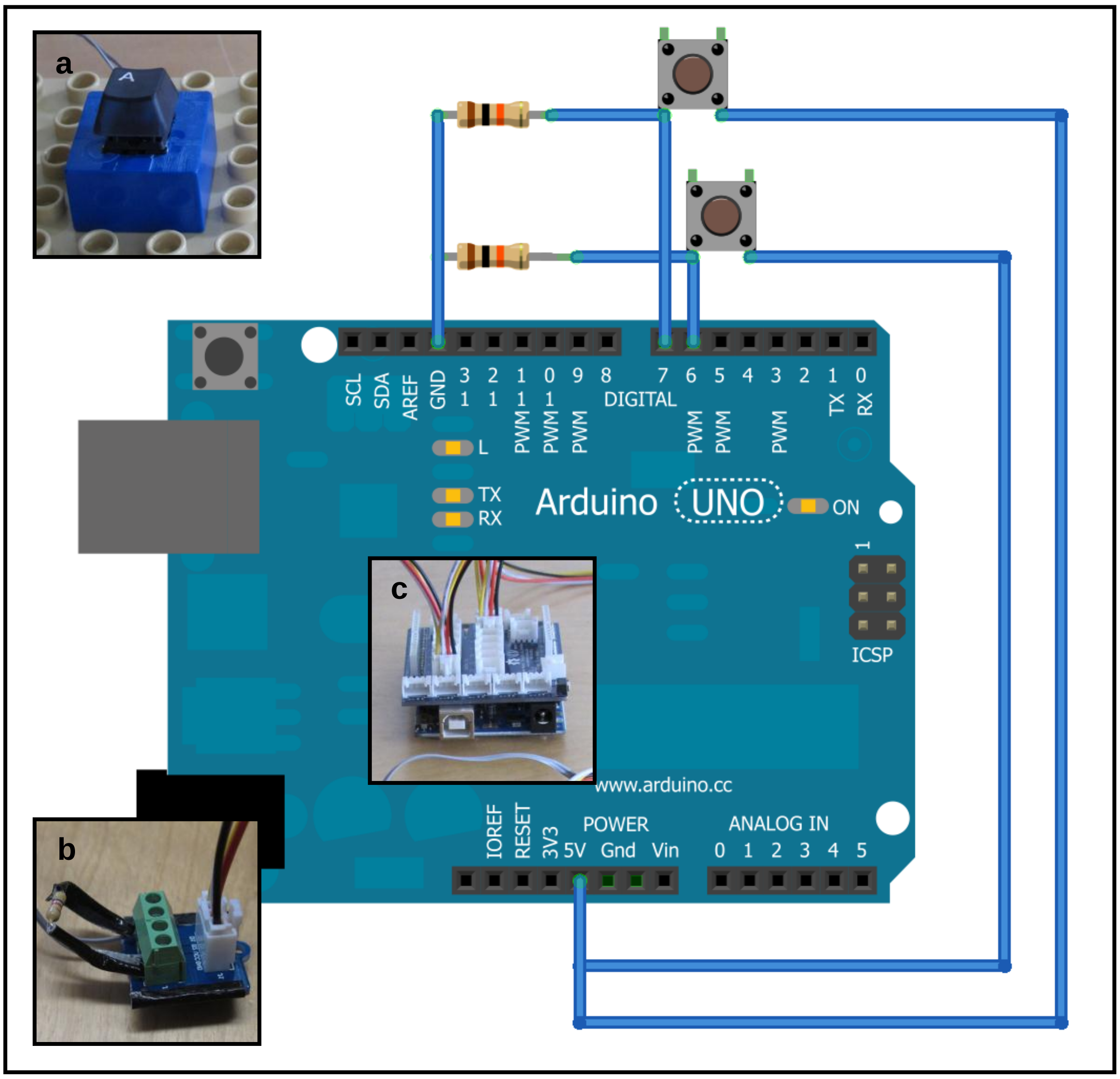
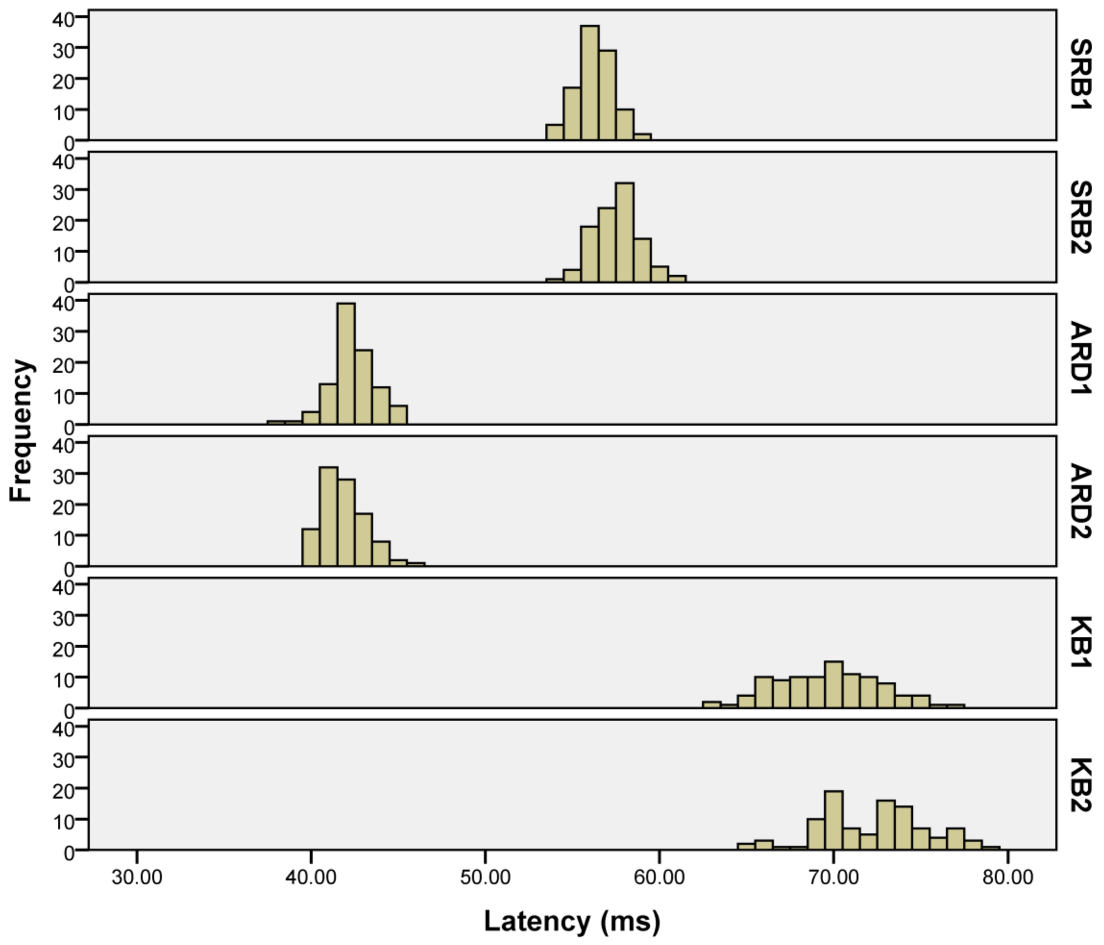


Figure 4

Histograms of measurements in Study 4. For each periphery, two measures with 100 trials each were taken. SRB = Serial Response Box, ARD = ARDUINO, KB = Keyboard.



Appendix

Checklist before collecting data

1. Is the same highest possible baud rate set in E-prime, Arduino sketch, and device manager?
2. Are all necessary columns added to the list containing the stimulus object or a higher order list?
3. Are the names of the stimulus object used in the final part of the E-Prime InLine adjusted to names actually used?
4. Does the E-Prime script leave a grace period of ca. 2 s after the start, before the first communication with the Arduino is attempted?
5. Is data logging turned on for the stimulus object?
6. Does the Arduino script define the correct pins for input and ouput?

Checklist after collecting data

7. Is the TrialNr (which is sent to the Arduino) the same as ArduinoTrialNr (which is received from the Arduino) for all trials?
8. Compute the difference *inlineRT – arduinoRT*, and check its distribution. If the average is large or the standard deviation is high, ArduinoRTs are going to be inaccurate or imprecise, respectively, as about 1/3 of the delay happens before ArduinoRT is measured.
9. Compute the difference *InlinePostSendingTime – InlineStartTime.* Trials with outlier differences may be unreliable, as the trial onset stimulus might have been delayed.

Listing 1: E-Prime InLine script

```
'============== PART 1 - SETUP ==============

'variables to save timestamps of start and end
'note: Double can hold values from 4.94066E-324 to 1.797693134862315E308
Dim startTime As Double
Dim postSendingTime As Double
Dim endTime As Double

'variables to be transmitted to Arduino
'note: Integer can hold values from -32768 to 32767
Dim trialNr As Integer
Dim trialDuration As Integer
Dim trialMessage As Integer

'generate random number as check number to verify correspondence of trial
'and what Arduino returns
trialNr = CInt(Random(0,32767))

'reads TrialDuration using the context object.
trialDuration = c.GetAttrib("TrialDuration")

'the following is just reserved, not currently used.
trialMessage = 999

'============== PART 2 - TALKING TO ARDUINO ==============

'construct and send String to Arduino.
'Arduino expects 3 Integers separated by a comma and ending
'with a non-integers sign other than a comma
'we End With a dot
Dim sendString As String
sendString = CStr(trialNr) & "," & CStr(trialDuration) & "," &
CStr(trialMessage) & "."

'remember when sent
startTime = Clock.ReadMicrosec

'send
Serial.WriteString sendString

'also save time after sending
postSendingTime = Clock.ReadMicrosec

'print them to debug
'uncomment following for debugging
'Debug.Print sendString

'============== PART 3 - LISTENING TO ARDUINO ==============

'now read and wait until the string terminator is in
Dim strData As String
Dim incoming As String
Dim nRead As Long

Dim waiting As Boolean
waiting = True

While waiting
       'read new data and add them to complete string
       nRead = Serial.ReadString(incoming)
       strData = strData & incoming
       'Anything come in yet?
       If Len(strData) > 0 Then
              'last sign of complete string is a . ?
```

```
                If Mid$(strData, Len(strData), 1) = "." Then
                        endTime = Clock.ReadMicrosec
                        waiting = False
                End If
        End If
Wend

'============== PART 4 - SAVING DATA ==============

'print them to debug
'uncomment following line for debugging
'Debug.Print strData

'disassemble string into integers, based on algorithm that is also used
'on the Arduino side
'here we expect 3 fields returning from Arduino: trialNr, RT, buttonStates
Dim NUMBER_OF_FIELDS As Integer
'how many comma separated fields we expect
NUMBER_OF_FIELDS = 3

'the current field being received
Dim fieldIndex As Integer
fieldIndex = 0

'array holding values for all the fields
Dim values(NUMBER_OF_FIELDS) As Long

Dim i As Integer
Dim ch As String
Dim chasc As Integer

For i = 1 To len(strData)
        ch = Mid$(strData, i, 1)
        chasc = Asc(ch)
        'is this an ascii digit between 0 and 9?
        If chasc >= 48 And chasc <= 57 Then
                'yes, accumulate the value
        values(fieldIndex) = (values(fieldIndex) * 10) + ch
        ' comma Is our separator, so move On To the Next field
        ElseIf ch = "," Then
                If fieldIndex < NUMBER_OF_FIELDS-1 Then
                fieldIndex = fieldIndex + 1    ' increment field index
                End If
     End If
Next i

'compute a check RT that is the difference between starting the inline
'and receiving the full Return String from arduino - should be longer
'than what RT says
Dim InlineRt As Double
InlineRT = endTime - startTime

'store generated random check number
c.Setattrib "TrialNr", trialNr

'uncomment following line for debugging
'Debug.Print values(0) & "\t" & values(1) & "\t" & values(2)

'save values returned from Arduino in three attributes of the StimulusList
c.SetAttrib "ArduinoTrialNr", values(0)
c.SetAttrib "ArduinoRT", values(1)
c.SetAttrib "ArduinoButtons", values(2)

'store timings measured by script
c.SetAttrib "InlineStartTime", startTime
c.SetAttrib "InlinePostSendingTime", postSendingTime
```

```
c.SetAttrib "InlineEndTime", endTime
c.Setattrib "InlineRT", InlineRT

'wait 100 ms to allow the trialStimulus object to finish
'uncomment if you expect reaction times less than the duration
'of the stimulus object
'sleep(100)

'save values also in .RT and .RESP, overwriting the no-response there

'CHANGE NAME OF STIMULUS OBJECT IN FOLLOWING LINES IF YOU
'USE A DIFFERENT ONE
       trialStimulus.RT = values(1)/1000
       trialStimulus.RESP = values(2)

       'now set ACC so that Feedback Display can evaluate it
       'this needs to be done manually here because trialStimulus
       'object is already finished,
       'And would not check the .RESP again to determine ACC
       If trialStimulus.CRESP = trialStimulus.RESP Then trialStimulus.ACC = 1
       If trialStimulus.CRESP <> trialStimulus.RESP Then trialStimulus.ACC = 0

'reset values to zero
For i = 0 To NUMBER_OF_FIELDS-1
       values(i) = 0
Next i
fieldIndex = 0
```

Listing 2: Arduino Sketch

```
#define led1 13
#define button1 6
#define button2 7
const int NUMBER_OF_FIELDS = 3; // how many comma separated fields we expect
int fieldIndex = 0;            // the current field being received
int values[NUMBER_OF_FIELDS];   // array holding values for all the fields

void setup()
{
  // Initialize serial port to same speed as set in e-prime
  Serial.begin(128000);
  pinMode(led1, OUTPUT);
  pinMode(button1, INPUT);
  pinMode(button2, INPUT);
}

void loop()
{
  readTrial();
}

void readTrial()
{
  if( Serial.available())
  {
    char ch = Serial.read();
    if(ch >= '0' && ch <= '9') // is this an ascii digit between 0 and 9?
    {
      // yes, accumulate the value
      values[fieldIndex] = (values[fieldIndex] * 10) + (ch - '0');
    }
    else if (ch == ',')  // comma is our separator
    {
      if(fieldIndex < NUMBER_OF_FIELDS-1)
        fieldIndex++;   // increment field index
    }
    else
    {
      // any character not a digit or comma ends the acquisition of fields
      runTrial(values[0], values[1], values[2]);
      //clear values again; otherwise they multiply up
      for (int i=0; i <= fieldIndex; i++)
      {
        values[i] = 0;
      }
      fieldIndex = 0;  // ready to start over
    }
  }
}

void runTrial(int trialNr, int trialDuration, int trialMessage)
{
  //uncomment following line to see board LED go on during trial
  //digitalWrite(led1, HIGH);

  unsigned long starttime = micros();
  unsigned long endtime = starttime + (unsigned long) trialDuration * 1000;
  unsigned long nowtime;
  int button1State = 0;
  int button2State = 0;

  do
  {
    button1State = digitalRead(button1);
    button2State = digitalRead(button2);
    nowtime = micros();
  } while (button1State == 0 && button2State == 0 && nowtime < endtime);

  //uncomment if you turned it on above
  //digitalWrite(led1, LOW);   // set the LED off again

  unsigned long rt = nowtime - starttime;
  //constructing answer string
  String buttonStates = "0"; // no response
  if (button1State == 1) {buttonStates = "1";}
  if (button2State == 1) {buttonStates = "2";}
  if (button1State == 1 && button2State == 1) {buttonStates = "3";}

  sendBack(trialNr, rt, buttonStates);
}

void sendBack(int trialNr, unsigned long rt, String message)
{
  Serial.print(trialNr,DEC);
  Serial.print(",");
  Serial.print(rt,DEC);
  Serial.print(",");
  Serial.print(message);
  Serial.print(".");
}
```