Behav Res (2016) 48:1591–1607
DOI 10.3758/s13428-015-0671-3

Tap Arduino: An Arduino microcontroller for low-latency auditory feedback in sensorimotor synchronization experiments

Benjamin G. Schultz[1] · **Floris T. van Vugt**[2]

Published online: 5 November 2015

Abstract Timing abilities are often measured by having participants tap their finger along with a metronome and presenting tap-triggered auditory feedback. These experiments predominantly use electronic percussion pads combined with software (e.g., FTAP or Max/MSP) that records responses and delivers auditory feedback. However, these setups involve unknown latencies between tap onset and auditory feedback and can sometimes miss responses or record multiple, superfluous responses for a single tap. These issues may distort measurements of tapping performance or affect the performance of the individual. We present an alternative setup using an Arduino microcontroller that addresses these issues and delivers low-latency auditory feedback. We validated our setup by having participants (N = 6) tap on a force-sensitive resistor pad connected to the Arduino and on an electronic percussion pad with various levels of force and tempi. The Arduino delivered auditory feedback through a pulse-width modulation (PWM) pin connected to a headphone jack or a wave shield component. The Arduino's PWM (M = 0.6 ms, SD = 0.3) and wave shield (M = 2.6 ms, SD = 0.3) demonstrated significantly lower auditory feedback latencies than the percussion pad (M = 9.1 ms, SD = 2.0), FTAP (M = 14.6 ms, SD = 2.8), and Max/MSP (M = 15.8 ms, SD = 3.4). The PWM and wave shield latencies were also significantly less variable than those from FTAP and Max/MSP. The Arduino missed significantly fewer taps, and recorded fewer superfluous responses, than the percussion pad. The Arduino captured all responses, whereas at lower tapping forces, the percussion pad missed more taps. Regardless of tapping force, the Arduino outperformed the percussion pad. Overall, the Arduino is a high-precision, low-latency, portable, and affordable tool for auditory experiments.

Keywords Auditory feedback · Sensorimotor synchronization · Motor timing · Musical Instrument Digital Interface (MIDI) · Microcontrollers

Humans show a remarkable capacity to align motor output with sensory input. For example, most individuals can effortlessly synchronize movements with the beat of music or the sound productions of a partner. In order to understand how synchrony is achieved, participants are asked to tap their finger along with metronomic stimuli and receive tap-triggered sounds (auditory feedback; cf. Repp, 2005; Repp & Su, 2013). These sensorimotor synchronization experiments present important methodological challenges: how can auditory feedback be presented at minimal latencies (ideally, within a few of milliseconds of the tap; see Aschersleben & Prinz, 1997), and how can tap times be collected reliably (i.e., without missing taps and with accurate millisecond timing information)? We compare standard methodologies to a novel solution using an Arduino microcontroller for use in sensorimotor synchronization experiments that require recording tapping responses and presenting auditory feedback.

Currently, several options exist for implementing sensorimotor synchronization experiments. Predominantly, studies have used musical instrument digital interface (MIDI) percussion pads (viz. drum pads) to trigger responses, computer software to record responses and control auditory feedback

✉ Benjamin G. Schultz
ben.schultz@maastrichtuniversity.nl;
benjamin.glenn.schultz@gmail.com

[1] International Laboratory for Brain, Music, and Sound Research, Université de Montréal, Département de psychologie, Montréal, Québec, Canada

[2] Department of Psychology, McGill University, Montreal, Quebec, Canada

(e.g., FTAP, Finney, 2001; Max/MSP, Cycling '74, 2014), and a tone generator to produce auditory feedback (cf. Repp, 2005). Common problems in studies using MIDI percussion pads are missing or superfluous responses (e.g., Mills, van der Steen, Schultz, & Keller, 2015; Pfordresher & Dalla Bella, 2011; Repp & Knoblich, 2007). A *missing* response occurs when a participant has tapped on the percussion pad but no response was recorded by the device. A *superfluous* response occurs when a participant has made a single tap on the percussion pad and multiple responses are recorded by the device. These situations become more problematic when auditory feedback is introduced because participants receive no feedback for a missing response and extra feedback for superfluous responses. Although some controllers allow the user to adjust the sensitivity of the drum pad and the threshold for what is considered to be a response, it is often difficult to obtain parameters that work for a range of response styles (i.e., from a soft through to a hard force of response). We compared the latencies of auditory feedback using the Arduino with other options that use a MIDI percussion pad to produce feedback through FTAP (Finney, 2001) or Max/MSP (Cycling '74, 2014).

The Arduino is a multipurpose, low-level microcontroller that is low-cost (i.e., less than USD 30), contains a processor that can receive analog and digital inputs, and can run programs written in a flavor of the C programming language. Here, we suggest that the Arduino provides the ideal infrastructure to implement tapping experiments because it can be purposed as a single-use device and bypass the hardware and software environments of standalone personal computers. Specifically, we have designed C codes and Python scripts to convert the Arduino into a sensorimotor synchronization measurement tool with the goal of collecting to-the-millisecond response times and producing low-latency auditory feedback. In addition, the C code provided here has parameters that aim to reduce the frequency of missing and additional responses regardless of the force of the response.

Several studies (e.g., D'Ausilio, 2012; Schubert, D'Ausilio, & Canto, 2013) have shown that the Arduino can record response latencies with less than 1-ms variability. The Arduino uses an internal clock that can record response times with microsecond precision. Using this clock, the Arduino can timestamp data at a high resolution and then send this data to a computer through USB. When exchanging data through a USB port, delays can be introduced by the polling speed, where the incoming information is only read periodically (125 Hz, or once every 8 ms, is the default for most operating systems, but some drivers are able to lower this polling speed). Since the timing information of responses is determined by the Arduino in real time, the polling speed of the USB is inconsequential to timing measurements and does not contribute additional error or variability. Moreover, the Arduino is capable of delivering auditory feedback directly through hardware (e.g., a headphone jack) thus removing any further delays introduced by USB communication. Therefore, the Arduino can be used to both collect data and produce auditory feedback at high resolutions. The C and Python codes we provide here can send the data from the Arduino either as a continuous time series that reads responses at every millisecond (1-kHz sample rate) or as response onset and offset times (with to-the-millisecond precision). Other systems that record timestamps in software after input is received through USB may have lower resolutions than systems that record timestamps internally (i.e., onboard timestamps), such as the Arduino.

Two MIDI-based software packages are commonly used for sensorimotor synchronization experiments: FTAP (Finney, 2001) and Max/MSP (Cycling '74, 2014). FTAP is a free, Linux-based software package that reports low latencies for providing auditory feedback when using MIDI devices. Max/MSP is a Windows and Mac compatible software package that is free to run, but requires purchasing a license to develop user-made scripts (e.g., experiments). We compared the latencies of auditory feedback produced by MIDI setups using FTAP and Max-MSP with those produced by the Arduino. We also measured the auditory feedback produced directly from a MIDI percussion pad to identify possible delays resulting from the device itself, although these were expected to be minimal due to reported specifications that MIDI devices take an average of 1 ms to send or receive a MIDI message (Casabona & Frederick, 1988). For the Arduino, we present two options for sound output: (1) an option were the audio output is a simple tone (sine wave or square wave) with a user-defined duration and pitch, produced through the Arduino's pulse-width modulation (PWM) pin (henceforth we refer to this option as PWM), and (2) an option for playing any wave file that has been saved on a secure digital card (SD card) through the Arduino wave shield (hereafter referred to as the wave shield). The first option requires less soldering expertise and hardware but auditory feedback is limited to simple sounds (e.g., pure tones and square waves). The second option is more expensive and requires more soldering expertise (see Adafruit, 2015), but allows the user to present any sound file. The C code for the Arduino, Python scripts for data collection (cross-platform), and instruction manuals for hardware and software are free to download (van Vugt & Schultz, 2015).

Experiment

We compared the performance of two Arduino-based feedback methods (i.e., PWM feedback and wave shield feedback) with two software-based feedback methods that interfaced with the MIDI percussion pad: one that used FTAP software and another that used Max/MSP software. In both cases, the software (FTAP or Max/MSP) generated tap-triggered sounds using a MIDI synthesizer (i.e., a tone generator). We tested these various configurations by conducting a common sensorimotor synchronization experiment in which participants had

to synchronize their responses to metronome clicks that occurred at periodic time intervals (cf. Repp, 2005). In order to establish the veridical onset times of responses and auditory feedback in each setup, we recorded data from the various devices simultaneously in a synchronized manner using an analog input box (AIB; BioSemi, Amsterdam, The Netherlands). The AIB recorded voltage readings from a force sensitive resistor (FSR) on which participants tapped and a vibration sensor (i.e., a piezo element) that measured tap-related vibrations. Participants did not hear auditory feedback for responses, but auditory feedback from the various devices (the Arduino, MIDI percussion pad, and MIDI sound module) was recorded by the AIB. The behavioral results of participants (e.g., synchrony with the metronome) are irrelevant to the aim of testing equipment performance and, therefore, are not reported.

Design and hypotheses

The dependent variable was the asynchrony between the response onset (i.e., taps, as measured by the FSR) and the audio onset (i.e., auditory feedback) recorded from each device. This asynchrony measured the latency of the auditory feedback for each source. Five sources of auditory feedback were measured: Arduino PWM, Arduino wave shield, percussion pad, FTAP, and Max/MSP. These sources of feedback could not all be measured simultaneously. Therefore, five conditions were used to measure various combinations of feedback sources, as shown in Table 1. In three conditions (see rows 1 to 3 in Table 1), participants tapped on an FSR that was placed on a drum pad of the percussion controller. In two conditions (see rows 4 and 5), participants only tapped on the drum pad of the percussion controller. This was done to ensure that the presence of the FSR did not hinder the percussion pad in terms of feedback latencies, missed responses, or superfluous responses; we compared the asynchronies of the onsets recorded by the piezo vibration sensor and the percussion pad audio onsets in the FSR present and absent conditions (see Table 1) to test whether the presence of the FSR increased the percussion pad audio latency. For missed and superfluous responses, we compared the responses recorded by the percussion pad in the FSR absent conditions with the responses recorded by the Arduino in the FSR present conditions.

Table 1 Arrangement of feedback conditions

Response Device(s)	Arduino	Software	FSR Presence
FSR and percussion pad	PWM	FTAP	Present
FSR and percussion pad	Wave shield	FTAP	Present
FSR and percussion pad	Wave shield	Max/MSP	Present
Percussion pad	None	FTAP	Absent
Percussion pad	None	Max/MSP	Absent

Each row represents one measurement condition in our experiment

Because individuals may differ in their tapping style and tapping force, we had six participants respond under three types of tapping force instructions: soft, moderate, and hard. To examine whether different tapping speeds affected feedback latency, number of missed taps, and number of double taps, a fast (240 beats per minute; bpm) and a slow (120 bpm) metronome rate were presented. Participants completed all conditions in a fully within-subjects design. We hypothesized that the Arduino conditions (PWM and wave shield) would demonstrate significantly lower latencies than the percussion pad, FTAP, and Max/MSP. Similarly, we hypothesized that the Arduino conditions would demonstrate significantly lower latency variability than the percussion pad, FTAP, and Max/MSP. Finally, we hypothesized that the Arduino would miss fewer valid responses and produce fewer superfluous responses than the percussion pad.

Method

Participants

The participants (N = 6) were four volunteers from the Université de Montréal and Concordia University, as well as the two experimenters. The participants had a mean age of 28.17 years (SD = 3.19, *range* = 23–32 years) and consisted of three females and three males.

Materials

Four computers were used for testing. The first computer (Intel Pentium 4, 3.00 GHz, running Windows XP) was used to record voltages and auditory signals though ActiView software (BioSemi, Amsterdam, The Netherlands). The second computer (Intel Xeon 5120, 1.86 GHz, running Windows XP) was used to present metronome stimuli (premade .wav files) and record the data from the Arduino via Python (v2.7). The third computer (Intel Core i7-2670QM, 2.2 GHz, running Linux Ubuntu v3.2.0-23 using the real-time kernel) was used to run FTAP. The fourth computer (MacBook Pro, Intel Core 2 Duo, 2.6GHz, running OS X v10.9.5) was used to produce auditory feedback through Max/MSP. Responses were recorded using a square FSR (3.81 cm, Interlink FSR 406) connected to an Arduino UNO R3 (see Fig. 1). The Arduino was powered via USB and also transmitted timing information though the serial USB port. In the PWM condition, the Arduino auditory feedback was delivered through a Sparkfun TRRS 3.5-mm jack breakout (BOB-11570; see Fig. 1), commonly known as a headphone jack (i.e., standard headphone or speakers could be connected to present the sounds to participants in an experimental setup). In the wave

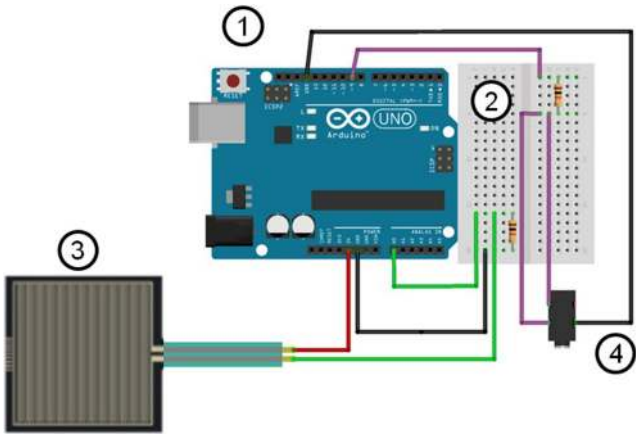

Fig. 1 Schematic wiring diagram for the Arduino PWM setup. The component numbers correspond to the (1) Arduino UNO, (2) breadboard, (3) square force-sensitive resistor (FSR), and (4) headphone jack (3.5 mm). Electric wires are indicated by black (grounds), red (power), green (FSR signals), and purple (audio signals). Both resistors are 10 kΩ. Headphone jack arrangements may vary, but the two purple wires connect to the *tip* and *ring 1* headphone inputs (left and right audio), and the ground (black) wire connects to the *ring 2* headphone input (see the datasheet of the headphone jack for input specifications). The wiring of the FSR is identical in the PWM and wave shield setups, and the wiring of the headphone jack is not necessary for the wave shield setup. This wiring diagram will allow prospective users to precisely reproduce our setup from the hardware components. The figure was created using the Fritzing software (Knörig, Wettach, & Cohen, 2009)

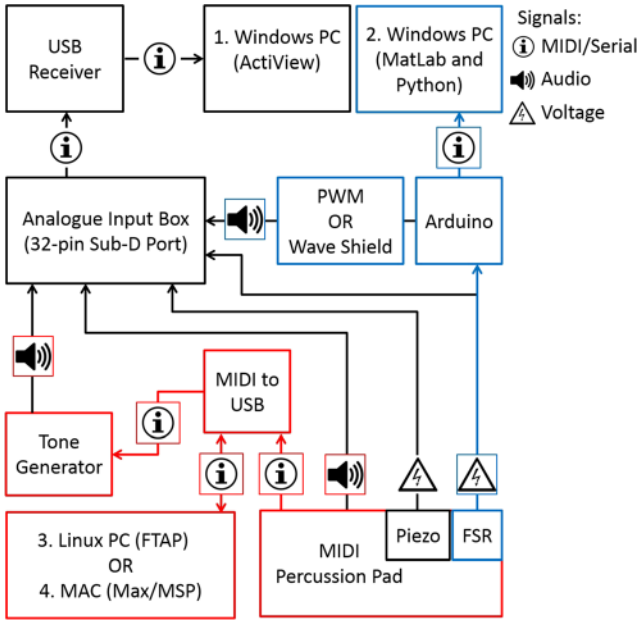

Fig. 2 Arrangement of devices used in the experiment. Note that this setup includes some devices that were used only in our validation experiment (see the black arrows and boxes) and are not necessary in a typical user setup. The red arrows and boxes represent the equipment that is used in a typical (non-Arduino) setup with FTAP or Max/MSP, included here for validation purposes. The blue arrows and boxes represent the equipment that is used in the Arduino setup described here. The primary node of the present validation experiment is the analog input box that records the inputs from the various devices synchronously (at 2048 Hz), so that latencies can be measured reliably. The computer numbers (1 to 4) match those referred to in the text

shield condition, the Arduino auditory feedback was delivered by the headphone jack of an Adafruit Wave Shield version 1.1 placed above the Arduino, with the FSR arranged in the same way as Fig. 1.

The auditory signal from each source was connected to the 32-pin Sub-D port (similar to a parallel port) of the AIB at a sampling rate of 2048 Hz[1] (see Fig. 2). Voltage changes caused by applying pressure to the FSR were simultaneously recorded by the AIB to synchronize participants' responses with the auditory feedback. The AIB has an analog-digital converter for each channel allowing the signals to be recorded synchronously. The FSR was placed on the bottom right drum pad of a Roland Handsonic HPD15 MIDI percussion pad. Voltages from the piezo vibration sensor placed on the same drum pad were obtained as a secondary measure of response onset time to test if the FSR increased the latencies of the percussion pad.[2] The audio output of the percussion pad was connected to the AIB to test the latency of audio being produced by the percussion pad itself. For the FTAP setup (see red boxes and arrows in Fig. 2), the MIDI signal from the percussion pad was connected to the PC through an M-Audio MIDIsport 2 × 2 (Anniversary Edition) USB-MIDI Interface. FTAP received the percussion pad MIDI signal via USB and sent the MIDI signal to the USB–MIDI interface, which then sent the MIDI signal to a Yamaha TX81Z MIDI synthesizer (i.e., tone generator) to produce the audio. The audio output from the tone generator was connected to the AIB to test the latency of audio being produced by FTAP. The conditions with Max/MSP were arranged in the same way as those with FTAP, using the tone generator to produce the audio, but with Max/MSP recording responses and controlling the auditory feedback (see the red boxes and arrows in Fig. 2).

Stimuli

The audio produced by the PWM and wave shield were 1046.5-Hz square waves of 20-ms duration. The audio produced by the percussion pad was the R13 snare drum, and the

[1] Note that 2048 Hz is not an acceptable sample rate for reproducing high-quality audio, but in this case we were simply using it to detect the onsets of auditory signals. The sampling rate is above the Nyquist frequency (double the frequency of interest) for the resolution at which we recorded responses (1000 Hz, i.e., to the millisecond), allowing us to detect asynchronies on the order of just below 0.5 ms.

[2] Fittingly, piezo elements are used in receiving responses from the HPD15 percussion pad (Smith, 2010). How these signals are filtered and mapped onto MIDI signals, however, is not specified.

audio produced by the synthesizer was the noise shot.[3] The percussion pad pitch was set to 1046.5 Hz, all effects and reverb were turned off, the trigger mode was set to "Shot" (short duration), the velocity curve was set to "Fixed16" (maximum volume for every trigger), pad sensitivity was set to 16 (maximum), the pad threshold was set to 1 (minimum), and mask time was set to 64 ms. In FTAP and Max/MSP, the MIDI frequency was set to MIDI note C6 (frequency = 1046.5 Hz), duration was set to 20 ms, and MIDI velocity was set to 127 (maximum).

Software

The C codes presented here perform a series of functions (van Vugt & Schultz, 2015). The Arduino's analog–digital converter (ADC) interprets FSR voltage changes as 10-bit integers ranging from 0 to 1023. First the Arduino reads the time stamp (in milliseconds) and the FSR voltage. If the FSR voltage is above our specified "ON" threshold (20 in 10-bit Arduino units; user definable), then the auditory feedback is played. Another sound is not produced until the FSR voltage decreases below our "OFF" threshold (10; user definable) for a user-specified amount of time (40 ms), and until a user-specified time after the onset (40 ms). These values were chosen to prevent double taps from arising when responding on the FSR and were arrived at (prior to conducting the experiment) from trial and error of attempting to induce auditory feedback without superfluous feedback or missed feedback. Lower voltage thresholds could be implemented in the Arduino code to increase the sensitivity, but the values used here indicated an optimal trade-off between high sensitivity and a low incidence of false alarms. When the offset is detected, the time stamp of onset, time stamp of offset, and the maximum FSR value are sent to the serial port in binary. The Python code runs on a separate PC and provides a graphical user interface (GUI) that collects data from the Arduino (through the USB) for further analysis (van Vugt & Schultz, 2015). In particular, the Python code reads binary data from the serial USB port and transforms the data into integers. These values are printed to a text file. The Python script records data until it is commanded to terminate (via closing the program, a set time value, or upon completion of a sound file). Note that it is not necessary to use this Python GUI to collect data from the Arduino: Users can write a custom script in any programming language that is capable of reading binary input from a serial USB port.

[3] We initially intended to use a square or sine wave of the same frequency (1046.5 Hz) for the MIDI patches on the percussion pad and tone generator, for comparability. Upon inspection of the audio signal, it was deemed that the attack times for the square- and sine-wave MIDI patches were slower, with less discernible onsets than some other patches. A nonexhaustive test of the available patches indicated that these two MIDI patches produced the fastest attack times and the most easily discernible onsets and offsets.

Our schematics and scripts are available online (van Vugt & Schultz, 2015), including detailed documentation, making this option accessible to those without much technical background. This repository shall be updated on the basis of suggestions from the community, and with the addition of scripts used in various experiments. The authors are willing to receive any questions about the hardware configuration and scripts to aid other researchers in using Arduino devices.

Procedure

Prior to any conditions that featured FTAP, the FTAP loop test was performed (see Finney, 2001). The FTAP loop consistently reported a 0.49-ms delay between output scheduling calls and that MIDI messages, on average, were sent and received within just over a millisecond (M = 1.01 ms, SD = 1.03 ms, *range* = 0 to 3 ms). Informed consent was obtained (CERAS-2014-15-/02-D). Participants were instructed to tap on the FSR that was placed on top of the percussion pad, or to tap in the center of the bottom right drum of the percussion pad. At the beginning of each trial, participants were instructed to tap with a soft, moderate, or hard force through text on a computer screen. These conditions were performed for all tempi (fast, slow) in a randomized order within each block, for five blocks. This procedure was repeated for all five feedback conditions (see Table 1; order counterbalanced across participants). At the end of the trial, participants were asked whether they had produced any double taps or had missed any responses after the first eight metronome ticks. If they responded "yes," the trial was repeated. Otherwise, they proceeded to the next trial. Participants were unable to monitor whether auditory feedback was being generated from any source. There were 48 metronome ticks per trial and, therefore, each trial had a 12-s (fast tempo) or 24-s (slow tempo) duration. Experiment sessions did not exceed 90 min. Participants were questioned regarding which tapping force was closest to their natural tapping force, and all six indicated that the moderate force was most natural.

Results

Onset extraction

Onsets of voltages and audio signal were detected from the traces recorded by the AIB using a custom-made MATLAB script. Onsets were detected as values that surpassed an amplitude threshold. The onset time was then established as the preceding point in time when the standard deviation (using ten sample windows) returned to baseline standard deviation levels (four times the median standard deviation of the trial). Detected audio onset times are shown in Appendix A. Missed responses were determined by examining the data output from the Arduino, FTAP, and Max/MSP and comparing them with

the number of expected responses (because participants were instructed to repeat the trial if any responses were missed). Only responses after the first eight metronome ticks were considered (i.e., 40 responses were expected per trial) and superfluous responses were first removed. Superfluous responses were measured as any response that occurred within 125 ms (half of the smallest interonset interval of the metronome) of another response.

Statistical analysis

As a result of missed responses, there were unequal numbers of data points for the asynchronies in different auditory feedback conditions. To deal with the problem of unequal data points, we fit a linear mixed-effects model (LMEM) that was able to cope with missing data, inhomogeneity of dependent variable variance across factor levels, and unbalanced designs. The LMEM was fit to the data with the fixed factors Signal (five levels: Arduino PWM, Arduino wave shield, percussion pad audio, FTAP, Max/MSP), Force (soft, medium, hard), and Tempo (fast, slow), and the random factors Participant (six levels) and Trial (five levels), where trial was nested within participant (i.e., we used the maximal random-effects structure justified by the experimental design, following Barr, Levy, Scheepers, & Tily, 2013). We further allowed unequal variances across the levels of the signal factor, which was decided on the basis of visual observation that the residuals were heterogeneous for the various signals, and also because some dependent variables (e.g., missed responses for the Arduino) had a standard deviation of zero. The model was fit using the *lme* function of the *nlme* library (Pinheiro, Bates, DebRoy, Sarkar, & R Development Core Team, 2015) for the R package of statistical computing (R Development Core Team, 2013), and unequal variance was implemented using the *varIdent* model formula term. Pair-wise contrasts were computed using generalized linear hypothesis testing for Tukey contrasts (corrected p values are reported), using the *glht* function in the *multcomp* library (Hothorn, Bretz, & Westfall, 2008). The LMEM was used to analyze all of our dependent variables (see Appendix B for the LMEM tables, and Appendix C for examples of the R code).

Classical null-hypothesis testing statistics are not designed to find evidence for the absence of a difference between conditions. Therefore, we calculated the Bayes factor to test that the FSR did not affect the performance (latency and variability) of the percussion pad in conditions in which the FSR was present as compared to when it was absent (see Table 1). To include conditions in which the FSR was absent, the asynchrony between the piezo vibration sensor onset and the audio onset of the percussion pad was compared between the FSR-present and -absent conditions. The Bayes factor quantifies the strength of evidence in favor of the null hypothesis (when less than 1) or in favor of the alternative hypothesis (when greater than 1; Rouder, Speckman, Sun, Morey, & Iverson, 2009). The Bayes factor was computed using the BayesFactor function in the BayesFactor library (Morey, Rouder, & Jamil, 2009).

FSR-aligned audio mean asynchrony

For asynchronies, we found significant main effects of signal and force (ps < .001), and no significant main effect of tempo (p = .11). All interactions reached significance (ps < .001). To test the hypothesis that audio onsets produced by the PWM and wave shield have lower latencies than other audio signals, pair-wise contrasts were conducted between signals. All signals were significantly different from one another (ps < .001) in the following order, from lowest to highest asynchrony: PWM, wave shield, percussion pad, FTAP, and Max/MSP (see Fig. 3). Pair-wise contrasts investigating the three-way interaction between signal, force, and tempo confirmed that the PWM, wave shield, and percussion pad were significantly different from each other and from FTAP and Max/MSP under all conditions. However, in some conditions FTAP and Max/MSP were not significantly different (e.g., soft × fast condition, p = 1.0), likely due to the high variability of the onset timings for these audio signals (see Fig. 3).

For the PWM and wave shield signals, there were significant differences between all force conditions (ps < .001), indicating that the asynchrony increased as force increased (see Fig. 4). For the percussion pad, FTAP, and Max/MSP signals, significant differences emerged between force conditions (ps < .02), but the asynchrony decreased as force increased. The PWM only demonstrated a significant difference between tempi for the soft force condition (p < .001), in which asynchrony was greater for the fast than for the slow tempo. The wave shield did not demonstrate significant differences between tempi for any force condition (ps > .83). The percussion pad and FTAP only demonstrated significant differences between tempi for the hard force (ps < .001); asynchronies for the fast tempo were greater than those for the slow tempo for the percussion pad, and the reverse trend was observed for FTAP. For Max/MSP, asynchronies were significantly greater for the slow than for the fast tempo for all force conditions (ps < .001).

FSR-aligned audio asynchrony variability

To compare the variability of the feedback signals, we computed the standard deviation of the asynchronies for each participant, signal, tempo, and force. These standard deviation estimates were corrected for biases arising from differences in the numbers of underlying data points using the c4 coefficient [see Eq. 1, where n = sample size, Γ = gamma function, $\Gamma(n) = (n-1)!$; see Cureton, 1968]. These variability estimates were then subjected to an LMEM with the same design as above, with asynchrony variability as the dependent variable. Trial was no longer included as a random effect, because at

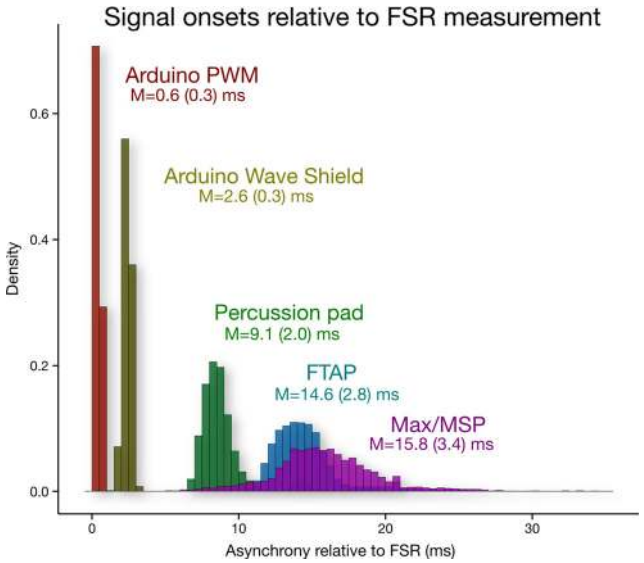

Fig. 3 Histogram of audio onset asynchronies relative to FSR onsets

most one data point was available for each combination of participant and condition.

$$c4(n) = \sqrt{\frac{2}{n-1}} * \frac{\Gamma\left(\frac{n}{2}\right)}{\Gamma\left(\frac{n-1}{2}\right)} \quad (1)$$

For asynchrony variability, we observed a significant main effect of signal ($p < .001$) and a significant interaction between signal and tempo ($p = .003$). No other main effects or interactions reached significance ($ps > .19$). To test the hypothesis that audio onsets produced by the PWM and wave shield have lower variability than the other audio signals, pair-wise contrasts were conducted between signals. Signal variabilities were generally significantly different from one another ($ps < .02$), with the exception of nonsignificant differences between the PWM and wave shield ($p = 1.0$) and between FTAP and Max/MSP ($p = .13$), and there were statistical trends for differences between the PWM and percussion pad ($p = .06$) and the wave shield and percussion pad ($p = .08$). As is shown in Fig. 5, the interaction between signal and tempo indicated that the percussion pad was significantly less variable than Max/MSP for the fast condition ($p < .01$) but not for the slow condition ($p = .23$). Moreover, a significant main effect of tempo emerged for Max/MSP ($p < .001$), indicating that the fast tempo was more variable than the slow tempo. It is possible that this tempo effect was driven by difficulties for Max/MSP to produce consistent timing of auditory feedback under high load—that is, when responses were more frequent—as compared to a slower response schedule. Overall, our results support the hypothesis that the PWM and wave shield provide less variable auditory feedback onsets than FTAP and Max/MSP. The PWM and wave shield showed near-significant trends for being less variable than the percussion pad.

FSR present versus absent comparison of percussion pad asynchronies

In this analysis, we included only the percussion pad audio asynchrony data, since the other signals (the tone generator output through FTAP or Max/MSP) occurred much later and were subject to additional temporal noise (probably due to the MIDI–USB and USB–MIDI conversions) and this noise is, by design, independent of whether an FSR was present or not. The LMEM was fit to the data with fixed factors Force, Tempo, and FSR Presence (two levels: present or absent; see

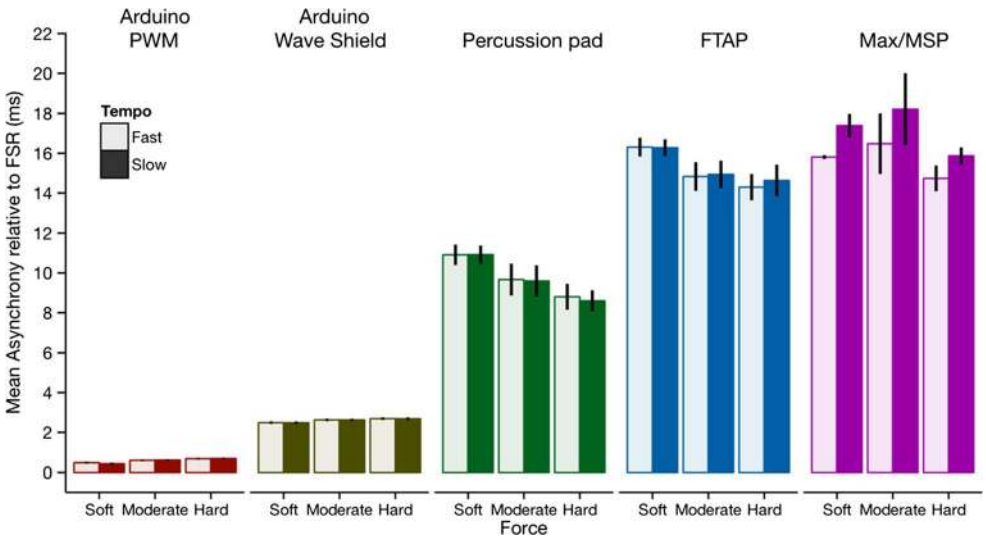

Fig. 4 Mean asynchrony relative to the FSR for audio signals in the tempo (fast, slow) and force (soft, moderate, hard) conditions. Whiskers represent standard errors of the means

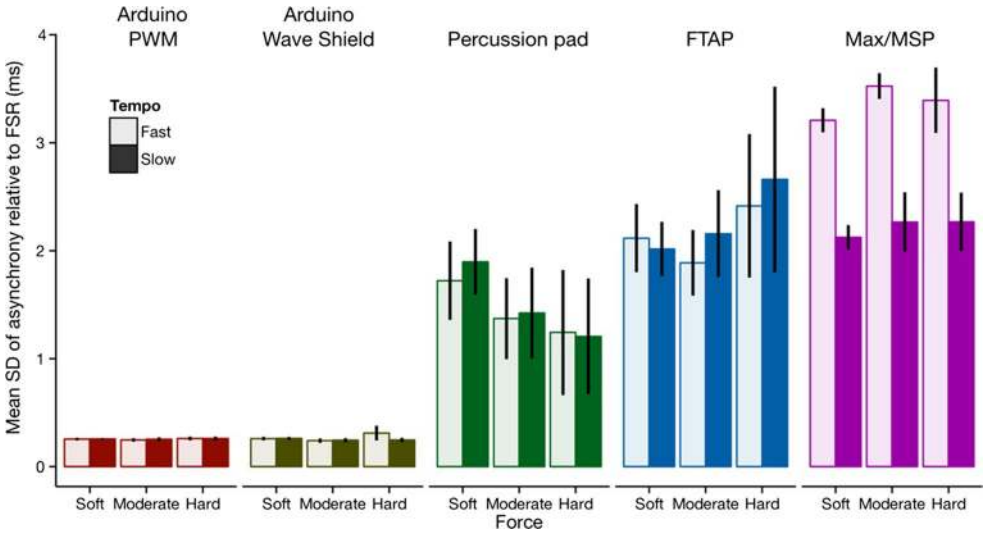

Fig. 5 Mean asynchrony variability (with standard deviations) for audio signals in the tempo (fast, slow) and force (soft, moderate, hard) conditions. Whiskers represent standard errors of the means

Table 1), and the random factors Participant (six levels) and Trial (five levels), where trial was nested within participant. The dependent variables were the asynchrony of the drum audio relative to the piezo vibration sensor onset (in milliseconds), and the variability of asynchrony.

All main effects and interaction effects reached significance (*p*s < .003), except for the three-way interaction between FSR presence, force, and tempo ($p = .60$). Tukey contrasts investigating the interaction between FSR presence and tempo revealed that percussion pad asynchronies demonstrated significantly lower latencies with the FSR present versus absent for the fast and slow tempi (*p*s < .001). Contrasts between FSR presence and force conditions demonstrated significantly lower latencies for the FSR-present than for the FSR-absent condition for hard and soft force (*p*s < .002), but not for moderate force ($p = .10$). These results indicate that the FSR presence generally *decreased* the asynchrony relative to conditions in which the FSR was absent. Since this indicates that the presence of the FSR produced a decrease in latencies, the Bayes factor was not calculated. The decreased latencies for FSR present as compared to absent may be attributed to the increased surface area provided by the FSR. The surface area of an adult human fingertip is approximately 2–3.2 cm^2 (Dandekar, Raju, & Srinivasan, 2003), and the square FSR has a surface area of 14.5 cm^2. The FSR may have spread out the tapping force over a larger area, thus improving the percussion pad's speed in detecting responses and, in turn, producing the audio signal more quickly.

The same analysis was conducted on the variances of the percussion pad audio asynchronies relative to the piezo vibration sensor, using the standard deviation of the onsets for each participant, trial, FSR presence condition, tempo, and force. We found no significant main effect of FSR presence and no significant interactions between FSR presence and force or FSR presence and tempo (*p*s > .29). We calculated the Bayes factor (*Bf*) to establish whether the FSR had no influence on the variability of the percussion pad. When we compared the model with participant as a random variable, there was evidence against including FSR presence in the model ($Bf = 0.001$), suggesting a low probability (odds = 1,000 to 1) that the presence of the FSR influenced the variability of the percussion pad.

Captured and superfluous responses

For the proportions of captured responses per trial (out of 40) and superfluous responses, we compared the numbers of responses captured by the Arduino in the FSR conditions with those by the percussion pad in the no-FSR conditions (see Table 1). The analysis on the proportions of captured responses demonstrated significant main effects of device (Arduino, percussion pad) and force (*p*s < .001), and a significant interaction between device and force ($p < .001$). The main effect for tempo and the other interactions did not reach significance (*p*s > .21). As is shown in Fig. 6, the Arduino recorded significantly more responses than the percussion pad in all force conditions (*p*s < .001; see Appendix D for the force profiles recorded by the Arduino). The percussion pad captured more responses in the moderate and hard force conditions than in the soft force condition (*p*s < .001), and the moderate and hard conditions did not significantly differ ($p = .41$).

For the number of superfluous responses, we observed main effects of device, force, and tempo (*p*s < .05) and a significant interaction between device and force ($p < .001$). The interaction between device and tempo approached significance ($p = .06$), but other interactions did not reach (*p*s > .20). As is shown in Fig. 7, the Arduino recorded significantly fewer superfluous responses than the percussion pad overall (*p*s < .001), but did not demonstrate significant differences when delineated by force condition

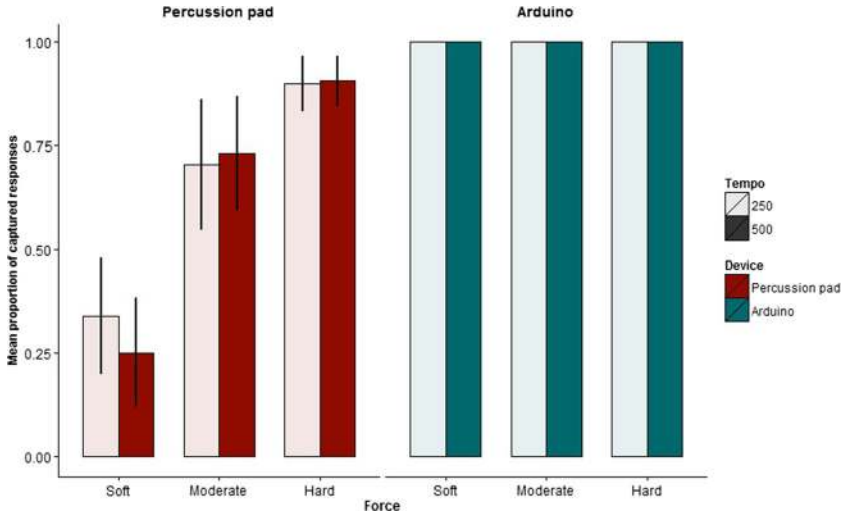

Fig. 6 Proportions of responses captured by the percussion pad and the Arduino in the force (soft, moderate, hard) and tempo (fast, slow) conditions. The Arduino registered 100 % of the produced taps, whereas the percussion pad missed taps, and more so when the tapping force was softer. Whiskers represent standard errors of the means

(*p*s > .11), possibly due to the high variability in the number of superfluous responses for the percussion pad. However, the Arduino recorded significantly fewer superfluous responses than the percussion pad for the slow tempo ($p = .002$), and demonstrated a statistical trend toward fewer superfluous responses for the fast tempo ($p = .05$).

Discussion

We demonstrated that the Arduino can be used as an effective way to implement sensorimotor synchronization experiments in which participants receive auditory feedback triggered by their taps. We validated the proposed setup by comparing the latencies and variability of the onset of auditory feedback and missed and superfluous recorded responses between the Arduino and two commonly used MIDI setups. The Arduino option was able to deliver auditory feedback with low latency and variability, which is considerably faster and less variable than the MIDI percussion pad, FTAP, and Max/MSP. Furthermore, the Arduino had fewer missed and superfluous responses than the percussion pad. These results, coupled with fact that the Arduino is less expensive than a MIDI percussion pad, make the Arduino a compelling option for sensorimotor synchronization experiments. We further showed that the

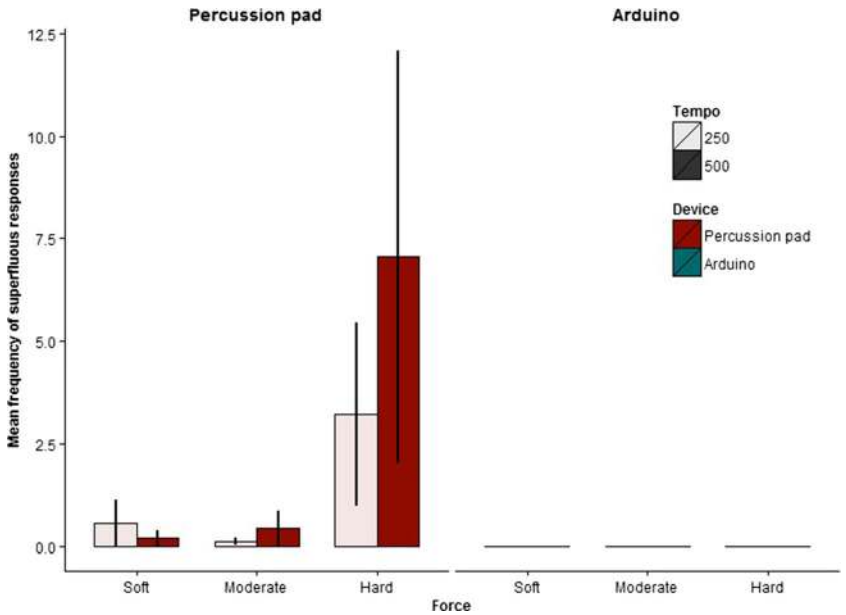

Fig. 7 Mean frequencies of superfluous responses captured by the percussion pad and Arduino in the soft, moderate, and hard force conditions for fast and slow tempi. Whiskers represent standard errors of the means

percussion pad was highly sensitive to varying tapping force levels; softer taps caused larger latencies and more missed taps. This poses an important problem for sensorimotor synchronization experiments using percussion pads, because participants may knowingly or unknowingly modulate their tapping force or tap in an unnatural manner to yield auditory feedback. All participants in our study reported that the moderate force was the most natural tapping force and might reflect the tapping force that participants in other experiments assume unless instructed otherwise. Our proposed Arduino setup was not affected by these force modulations, making it a more reliable instrument to measure sensorimotor tapping responses. Moreover, the Arduino allows the flexibility to tailor the parameters to the expected timeframe and force of responses.

There are several reasons larger latencies would have emerged in the MIDI-based setups (FTAP and Max/MSP). The percussion pad itself must detect responses using real-time signal processing to record responses and tap forces. Percussion pad manufacturers do not release the signal processing algorithms to the consumer so it is difficult to divine precisely how this is performed. It is also difficult to know when the MIDI signal is sent through the MIDI out port relative to the production of the audio on board the percussion pad. If one assumes that the MIDI signal and percussion pad audio are produced somewhat synchronously, then it appears that the percussion pad is accountable for the majority of the latency (see Fig. 3). The other sources of latency include the MIDI–USB conversion (and vice versa), the computer processing of the MIDI inputs and outputs, and the generation of the audio with the tone generator. FTAP and Max/MSP might actually contribute negligibly to the latencies of auditory feedback. However, the number of separate devices and connections that are required to implement these setups increases the latency of auditory feedback and is unavoidable for interfacing MIDI devices with FTAP and Max/MSP.[4] The benefit of using the Arduino is fewer connections between the devices that record responses and generate auditory feedback. Moreover, the performance of the Arduino is completely independent of the computer that is reading data from the Arduino, increasing reproducibility between different labs and experiments.

We acknowledge that other computer systems and hardware configurations might decrease the latencies observed in FTAP and Max/MSP—for example, by using a conventional peripheral component interconnect (PCI) MIDI sound card (see Nelson & Thom, 2004). Such configurations, however, would neither circumvent the latencies and variability introduced by the percussion pad, nor decrease the number of missed and superfluous responses resulting from the percussion pad. Furthermore, none of the published articles that have used FTAP of Max/MSP have reported using configurations that opt for a PCI MIDI sound card or the use of a joystick controller port[5] (i.e., a serial game port, as suggested in Finney, 2001). Other MIDI percussion pads may not produce as many missed responses but some papers have reported unrecorded responses with other devices (e.g., Pfordresher & Dalla Bella, 2011; Repp & Knoblich, 2007). As the present study shows, the Tap Arduino setup detected 100 % of taps and produced a total of two superfluous taps throughout the experiment. Therefore, we demonstrated that the Tap Arduino is a reliable tool for recording responses.

Although there are other software (e.g., MatTAP; Elliott, Welchman, & Wing, 2009) and hardware (e.g., button boxes as used in Snyder et al., 2006) options, the latencies and variability of these alternatives are often untested or unreported. Here, we tested two of the most common configurations using MIDI controllers; other options generally require external devices (e.g., data acquisition cards, as in Elliott et al., 2009) that are more expensive than the Arduino configurations described here. The cost of the Arduino microcontroller and associated equipment is a fraction of the cost of most MIDI percussion controllers and MIDI samplers that do the same task. The total cost of the PWM setup is approximately USD 65.00, and the wave shield setup costs approximately USD 110.00. This is can be compared to the MIDI percussion controllers (and dependent devices such as MIDI samplers and MIDI-to-USB cables), which can cost anywhere from USD 600.00 to over USD 1,500.00 for a full system.

One issue that has not been addressed is how best to synchronize the timing of responses with an external auditory stimulus (e.g., a metronome pacing sequence). Although other systems claim high timing resolutions for synchronizing responses with external stimuli (e.g., StimSync, Rorden & Hanayik, 2014; MatTAP, Elliott et al., 2009) many other commercially available setups are not subjected to peer-review and the veridical

[4] Max/MSP also provides a virtual MIDI synthesizer that can produce auditory feedback through the computer's audio and headphone ports. We attempted to test the virtual MIDI synthesizer in Max/MSP but the latency in auditory feedback was noticeably larger and, therefore, we proceeded to only test the arrangement reported here.

[5] Steve Finney (e.g., Finney, 2001) and Peter Pfordresher (Pfordresher, personal communication, June 23, 2015) have used the joystick controller port in their experiments that use FTAP and, although this is not specified, other studies may have used a similar MIDI interface device.

response-stimulus asynchronies associated with such software packages are unknown. We have included a beta script in our software package for syncing an auditory wave (.wav) file with Arduino responses ("TapArduinoSound.py") but the actual asynchrony between the Arduino responses and onset of computer-generated audio remains to be tested on multiple systems. This is a problem for experiment setups in general and, until this matter is resolved, the expensive options such as data acquisition cards and AIBs remain the most temporally precise methods for synchronizing responses and stimuli.

There are some limitations of the Tap Arduino package. First, unlike FTAP and Max/MSP, Tap Arduino cannot interface with MIDI devices such as piano keyboards. Second, the arrangement of the Tap Arduino presented here is incompatible with the MIDI protocol and, therefore, cannot take advantage of the library of MIDI sounds. Third, the Tap Arduino cannot dynamically change the intensity of auditory feedback as a result of changes in tapping force. However, a strength of the Tap Arduino package is that is can play any sound that can fit on an SD card as a wave file. We have also included codes that can alter auditory feedback in terms of temporal delay (i.e., delayed auditory feedback), frequency (i.e., pitch), timbre, and intensity (i.e., loudness). Another benefit of the Arduino microcontroller more generally is that it is expandable and can be programmed to communicate with a large range of devices that read serial protocol. A user is not limited to using an FSR as used in the present study but may, instead, use a piezo element to record tap vibrations, a circular potentiometer for circle drawing, or a simple button similar to a computer keyboard key (see Schubert et al., 2013). The drum pads used in videogames, such as Rock Band, and the percussion pad tested here use the piezo elements to record onsets. Through the Arduino, it is possible to have fine control over the thresholds and sensitivity that allows onsets to trigger auditory feedback.

Now that we have benchmark measurements for the latencies and variability of feedback using the Arduino and MIDI options, future research could determine the implications of having delayed or variable feedback in behavioral experiments. Aschersleben and Prinz (1997) have shown that increasing the latency of auditory feedback as much as 30 ms can increase the mean negative asynchrony of responses (relative to metronome ticks) from –20 ms to less than –40 ms. These results indicate that unwanted delays in auditory feedback (resulting from the experimental hardware used) influence behavior in sensorimotor synchronization experiments. The Tap Arduino circumvents this problem by presenting auditory feedback within milliseconds and could therefore be used to find the threshold at which behavior is influenced by delayed auditory feedback. It is possible that delays observed in FTAP and Max/MSP are inconsequential for performance in sensorimotor synchronization experiments, particularly given that people may adapt to them (Aschersleben & Prinz, 1997). However, it is likely that the variability would make it difficult to habituate to delays in auditory feedback, an assertion that is yet to be tested empirically. Similarly, the impact of missing and superfluous responses in experiments that present auditory feedback for pairs or individuals in sensorimotor synchronization remains unknown. This could be investigated using the Tap Arduino package that is sensitive enough not to miss responses and frugal enough not to record superfluous responses.

Conclusion

We have presented C codes and Python scripts for using an Arduino microcontroller as a tool for measuring responses at high resolutions and presenting low-latency auditory feedback in sensorimotor synchronization experiments. The Arduino was able to collect responses with high precision (i.e., without missing responses) while minimizing false alarms (i.e., superfluous responses). Our codes, scripts, and hardware instructions are freely available online (van Vugt & Schultz, 2015). The PWM auditory feedback option is faster and requires purchasing less hardware, but can only present simple sounds such as pure tones or square waves. The wave shield auditory feedback option allows the presentation of any sound file, but it has a slightly higher latency, is more expensive, and it requires a higher level of soldering ability (see Adafruit, 2015). Both of the Arduino options demonstrated lower, and less variable, auditory feedback latencies than FTAP and Max/MSP. On the basis of these results, we suggest that the Tap Arduino provides powerful tools for sensorimotor synchronization experiments, because it is highly precise and resistant to false alarms, produces low-latency feedback, and is portable and more affordable than existing solutions.

Acknowledgments The authors thank Marcello Wanderley for use of the HPD15 percussion pad, Joseph Malloch for use of the Yamaha TX81Z synthesizer, James O'Callaghan for aiding in the construction of the Max/MSP script, Alexander Demos for comments on the experiment design and analysis, and the participants who volunteered their time.

Appendix A

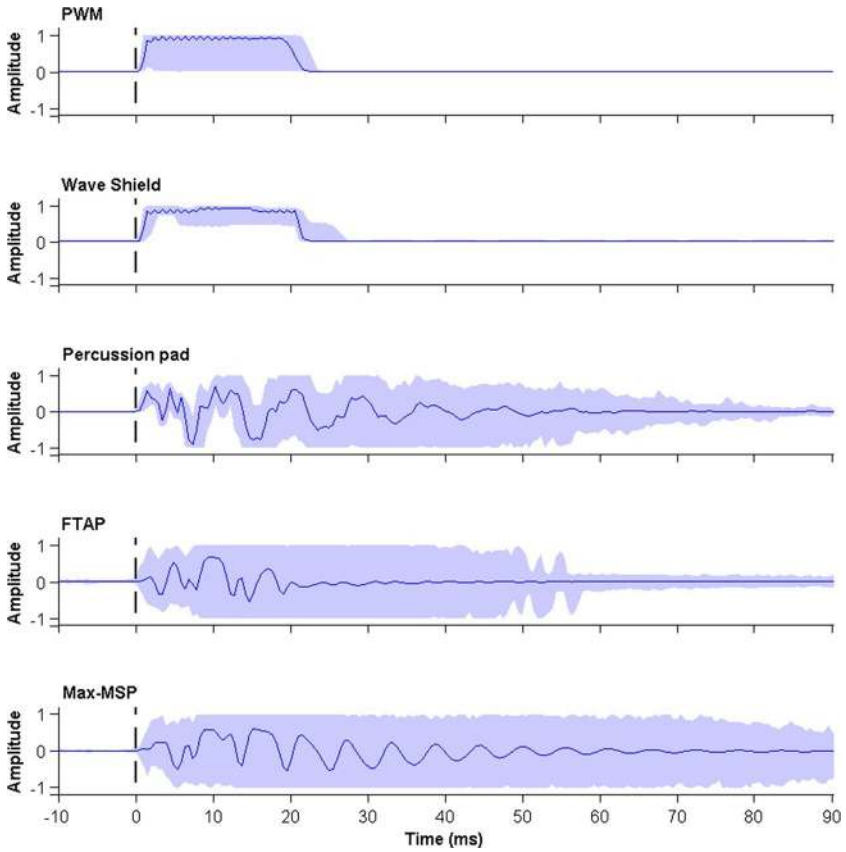

Fig. 8 Audio onsets detected by the custom-made MATLAB script. The vertical dashed lines represent the detected time of onset (zero). The solid blue lines represent the mean amplitude of each signal over time (in all conditions), and the shaded regions represent the minimum and maximum amplitudes of each signal

Appendix B

Table 2 Linear mixed-effects model for auditory feedback asynchronies, and asynchrony variability, relative to the force sensitive resistor (FSR)

	FSR Asynchronies		FSR Asynchrony Variability	
	Estimate	*SE*	Estimate	*SE*
Fixed Effects				
(Intercept)	0.68***	0.04	0.26	0.33
Signal: Arduino wave shield	2.01***	0.01	0.05	0.40
Signal: Percussion pad	8.01***	0.02	0.98*	0.40
Signal: FTAP	13.45***	0.04	2.16***	0.40
Signal: Max/MSP	13.92***	0.10	3.07***	0.42
Force: Moderate	–0.08***	0.01	–0.01	0.40
Force: Soft	–0.20***	0.01	–0.01	0.40
Tempo: 120 BPM	0.01	0.01	0.00	0.40
Signal: Arduino Wave Shield × Force: Moderate	0.016	0.01	–0.06	0.57
Signal: Percussion Pad × Force: Moderate	0.58***	0.04	0.14	0.57
Signal: FTAP × Force: Moderate	0.39***	0.06	–0.52	0.57
Signal: Max/MSP × Force: Moderate	1.07***	0.15	0.15	0.60
Signal: Arduino Wave Shield × Force: Soft	0.001	0.01	–0.05	0.57
Signal: Percussion pad × Force: Soft	1.77***	0.05	0.49	0.57
Signal: FTAP × Force: Soft	1.77***	0.08	–0.29	0.57
Signal: Max/MSP × Force: Soft	1.38***	0.25	–0.04	0.65
Signal: Arduino Wave Shield × Tempo: 120 BPM	–0.01	0.01	–0.06	0.57
Signal: Percussion pad × Tempo: 120 BPM	–0.14**	0.03	–0.04	0.57
Signal: FTAP × Tempo: 120 BPM	0.30***	0.06	0.24	0.57
Signal: Max/MSP × Tempo: 120 BPM	1.27***	0.14	–1.13	0.60
Force: Moderate × Tempo: 120 BPM	0.004	0.01	0.01	0.57
Force: Soft × Tempo: 120 BPM	–0.06***	0.01	0.00	0.57
Signal: Arduino Wave Shield × Force: Moderate × Tempo: 120 BPM	0.003	0.01	0.06	0.81
Signal: Percussion pad × Force: Moderate × Tempo: 120 BPM	0.14**	0.05	0.08	0.81
Signal: FTAP × Force: Moderate × Tempo: 120 BPM	–0.16*	0.08	0.02	0.81
Signal: Max/MSP × Force: Moderate × Tempo: 120 BPM	0.06	0.21	–0.14	0.85
Signal: Arduino Wave Shield × Force: Soft × Tempo: 120 BPM	0.06***	0.01	0.06	0.81
Signal: Percussion pad × Force: Soft × Tempo: 120 BPM	0.28***	0.08	0.21	0.81
Signal: FTAP × Force: Soft × Tempo: 120 BPM	–0.25*	0.12	–0.35	0.81
Signal: Max/MSP × Force: Soft × Tempo: 120 BPM	0.46	0.35	0.04	0.92
Random Factors				
Participant	0.10		0.35	
Trial (within Participant)	0.01		N/A	
Residual	0.28		0.64	
Goodness of Fit				
Log Likelihood	–129,155.6		–170.9	
AIC	258,385.2		405.7	
BIC	258,735.5		506.1	

*** $p < .001$, ** $p < .01$, * $p < .05$

Table 3 Linear mixed-effects model for auditory feedback asynchronies, and asynchrony variability, relative to the Piezo vibration sensor

	Piezo Asynchronies		Piezo Asynchrony Variability	
	Estimate	*SE*	Estimate	*SE*
Fixed Effects				
(Intercept)	8.09***	1.15	2.05**	0.72
Force: Moderate	1.69***	0.05	0.76	0.61
Force: Soft	4.66***	0.08	1.13	0.61
Tempo: 120 BPM	–0.42***	0.05	–0.40	0.61
FSR: absent	0.28***	0.07	–0.37	0.61
Force: Moderate × Tempo: 120 BPM	0.14	0.07	0.12	0.86
Force: Soft × Tempo: 120 BPM	0.72***	0.11	0.44	0.86
Force: Moderate × FSR: absent	–0.26*	0.10	–0.17	0.88
Force: Soft × FSR: absent	0.22	0.15	0.41	0.88
Tempo: 120 BPM × FSR: absent	0.38***	0.09	0.44	0.86
Force: Moderate × Tempo: 120 BPM × FSR: absent	–0.01	0.14	–0.27	1.23
Force: Soft × Tempo: 120 BPM × FSR: absent	–0.22	0.22	–0.58	1.24
Random Factors	*SD* Estimate		*SD* Estimate	
Participant	2.81		1.30	
Residual	2.83		0.96	
Goodness of Fit				
Log Likelihood	–94,096.58		–104.03	
AIC	188,221.20		236.06	
BIC	188,340.90		267.33	

*** $p < .001$, ** $p < .01$, * $p < .05$

Table 4 Linear mixed-effects model for the proportion of captured responses and number of superfluous responses

	Proportion of Captured Responses		Superfluous Responses	
	Estimate	*SE*	Estimate	*SE*
Fixed Effects				
(Intercept)	1.00***	0.03	0.00	1.60
Device: Percussion Pad	–0.11***	0.03	3.22	2.19
Force: Moderate	0.00	0.02	0.00	2.19
Force: Soft	0.00	0.02	0.00	2.19
Tempo: 120 BPM	0.00	0.02	0.01	2.19
Device: Percussion Pad × Force: Moderate	–0.21***	0.04	–3.10	3.10
Device: Percussion Pad × Force: Soft	–0.61***	0.04	–2.65	3.10
Device: Percussion Pad × Tempo: 120 BPM	0.01	0.04	3.83	3.10
Force: Moderate × Tempo: 120 BPM	0.00	0.03	–0.01	3.10
Force: Soft × Tempo: 120 BPM	0.00	0.03	0.00	3.10
Device: Percussion Pad × Force: Moderate × Tempo: 120 BPM	0.02	0.05	–3.51	4.38
Device: Percussion Pad × Force: Soft × Tempo: 120 BPM	–0.09	0.05	–4.20	4.38
Random Factors	*SD* Estimate		*SD* Estimate	
Participant	0.05		0.90	
Residual	0.17		3.46	
Goodness of Fit				
Log likelihood	404.67		–193.33	
AIC	–781.34		414.67	
BIC	–710.08		446.54	

*** $p < .001$, ** $p < .01$, * $p < .05$

Appendix C: R code for linear mixed-effects model

```
# Linear mixed-effects model
fsr.lme <- lme(Asynchrony ~ Signal*Force*Tempo, random = ~1|Participant/Trial,
          data = fsr_data, method = "ML", weights = varIdent(form = ~1|Signal))

# Check estimates and coefficients
summary(fsr.lme)

# View main effects and interactions
# (note: this function is NOT an analysis of variance but is
# an analogous function for LMEM)
anova(fsr.lme)

# Example: Planned comparisons for main effect of Signal
summary(glht(fsr.lme,
        linfct = mcp(Signal = "Tukey")))
# Above, "Signal" can be replaced by "Force" or "Tempo" to obtain planned comparisons

# Example: Interactions between Signal and Force
fsr_data$SigFor <- factor(interaction(fsr_data$Signal,fsr_data$Force))

fsr.tmp.lme <- lme(Asynchrony ~SigFor*Tempo, random = ~1|Participant/Trial,
                data = fsr_data, method = "ML", weights = varIdent(form = ~1|Signal))

summary(glht(fsr.tmp.lme,linfct = mcp(SigFor = "Tukey")))
# Above, "Signal" and "Force" can be swapped with "Tempo" to obtain planned comparisons

# Three-way interactions
fsr_data$SigforTemp <- interaction(fsr_data$Signal,fsr_data$Force, fsr_data$Tempo)

fsr.comb.lme <- lme(Asynchrony~SigForTemp, random = ~1|Participant/Trial,
                 data = fsr_data, method = "ML", weights = varIdent(form = ~1|Signal))

summary(glht(fsr.comb.lme,linfct = mcp(SigForTemp = "Tukey")))
```

Appendix D

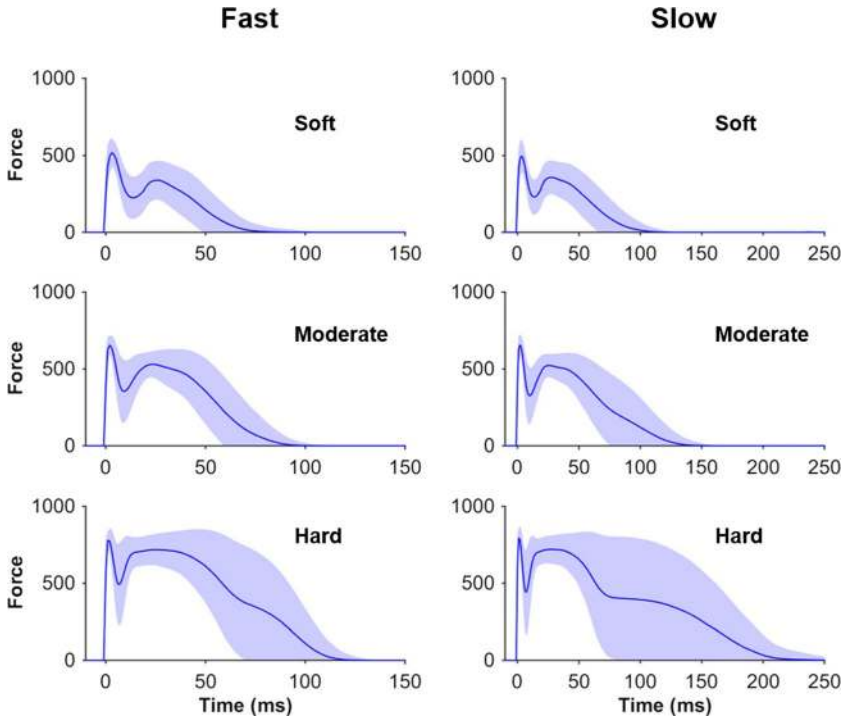

Fig. 9 Mean force values recorded from by the Arduino for the fast and slow tempi under conditions of soft, moderate, and hard tapping instructions. The unit of force is a 10-bit integer ranging from 0 to 1023, representing the resistance to the force placed on the FSR. The shaded areas represent the standard deviations

Arduino force analysis

The maximum force value between the onset and offset of the tap was recorded for each tap. The mean maximum force value for each trial was recorded and analyzed in a 2 (tempo) by 3 (force) repeated measures analysis of variance with Participant and Trial as random factors. We found a significant main effect of force [$F(2, 10) = 82.42$, $p < .001$, $\eta_p^2 = .94$], indicating that the hard instruction ($M = 782.2$, $SD = 82.37$) resulted in a significantly greater force than did the soft ($M = 501.1$, $SD = 99.20$; $p < .001$) and moderate ($M = 665.0$, $SD = 77.67$; $p < .001$) instructions, and the moderate instruction resulted in a significantly greater force than the soft instruction ($p < .001$). There was no significant main effect of tempo ($p = .64$), and the force by tempo interaction approached significance ($p = .06$). The near-significant interaction between force and tempo reflected that the slow tempo resulted in significantly greater force than the fast tempo for the hard force instruction ($p < .001$), but not for the soft or moderate force instructions ($ps > .27$)

References

Adafruit. (2015). Adafruit wave shield for Arduino kit. Retrieved 19 June, 2015, from www.adafruit.com/product/94

Aschersleben, G., & Prinz, W. (1997). Delayed auditory feedback in synchronization. *Journal of Motor Behavior, 29,* 35–46.

Barr, D. J., Levy, R., Scheepers, C., & Tily, H. J. (2013). Random effects structure for confirmatory hypothesis testing: keep it maximal. *Journal of Memory and Language, 68,* 255–278. doi:10.1016/j.jml.2012.11.001

Casabona, H., & Frederick, D. (1988). *Advanced MIDI applications*. New York: Alfred Music.

Cureton, E. E. (1968). Unbiased estimation of the standard deviation. *The American Statistician, 22,* 22.

Cycling '74. (2014). Max/MSP 7.0, jitter 1.2.3 graphical audio and video environment [Computer program]. Retrieved 12 August, 2014, from www.cycling74.com

D'Ausilio, A. (2012). Arduino: a low-cost multipurpose lab equipment. *Behavior Research Methods, 44,* 305–313. doi:10.3758/s13428-011-0163-z

Dandekar, K., Raju, B. I., & Srinivasan, M. A. (2003). 3-D finite-element models of human and monkey fingertips to investigate the mechanics of tactile sense. *Journal of Biomechanical Engineering, 125,* 682–691.

Development Core Team, R. (2013). *R: A language and environment for statistical computing*. Vienna: R Foundation for Statistical Computing. **Retrieved from www.R-project.org/**

Elliott, M. T., Welchman, A. E., & Wing, A. M. (2009). MatTAP: a MATLAB toolbox for the control and analysis of movement synchronisation experiments. *Journal of Neuroscience Methods, 177,* 250–257.

Finney, S. A. (2001). FTAP: a Linux-based program for tapping and music experiments. *Behavior Research Methods, Instruments, & Computers, 33,* 65–72.

Hothorn, T., Bretz, F., & Westfall, P. (2008). Simultaneous inference in general parametric models. *Biometrical Journal, 50,* 346–363.

Knörig, A., Wettach, R., & Cohen, J. (2009). Fritzing: A tool for advancing electronic prototyping for designers. In *Proceedings of the 3rd International Conference on Tangible and Embedded Interaction* (pp. 351–358). New York: ACM Press.

Mills, P. F., van der Steen, M. C., Schultz, B. G., & Keller, P. E. (2015). Individual differences in temporal anticipation and adaptation during sensorimotor synchronization. *Timing & Time Perception, 3,* 13–31. doi:10.1163/22134468-03002040

Morey, R. D., Rouder, J. N., & Jamil, T. (2009). BayesFactor: An R package for Bayesian data analysis (R package version 09.10-2). Retrieved from http://bayesfactorpcl.r-forge.r-project.org/

Nelson, M., & Thom, B. (2004). A survey of real-time MIDI performance. In *Proceedings of the 2004 conference on New interfaces for musical expression* (pp. 35–38). Singapore: National University of Singapore.

Pfordresher, P. Q., & Dalla Bella, S. (2011). Delayed auditory feedback and movement. *Journal of Experimental Psychology: Human Perception and Performance, 37,* 566–579.

Pinheiro, J., Bates, D., DebRoy, S., Sarkar, D., & R Development Core Team. (2015). nlme: Linear and nonlinear mixed effects models (R package version 3.1-120). Retrieved from http://CRAN.R-project.org/package=nlme

Repp, B. H. (2005). Sensorimotor synchronization: a review of the tapping literature. *Psychonomic Bulletin & Review, 12,* 969–992. doi:10.3758/BF03206433

Repp, B. H., & Knoblich, G. (2007). Toward a psychophysics of agency: detecting gain and loss of control over auditory action effects. *Journal of Experimental Psychology: Human Perception and Performance, 33,* 469–482. doi:10.1037/0096-1523.33.2.469

Repp, B. H., & Su, Y. H. (2013). Sensorimotor synchronization: a review of recent research (2006–2012). *Psychonomic Bulletin & Review, 20,* 403–452. doi:10.3758/s13423-012-0371-2

Rorden, C., & Hanayik, T. (2014). StimSync: open-source hardware for behavioral and MRI experiments. *Journal of Neuroscience Methods, 227,* 90–99.

Rouder, J. N., Speckman, P. L., Sun, D., Morey, R. D., & Iverson, G. (2009). Bayesian *t* tests for accepting and rejecting the null hypothesis. *Psychonomic Bulletin & Review, 16,* 225–237. doi:10.3758/PBR.16.2.225

Schubert, T. W., D'Ausilio, A., & Canto, R. (2013). Using Arduino microcontroller boards to measure response latencies. *Behavior Research Methods, 45,* 1332–1346.

Smith, F. (2010, November 20). Inside a drum synth/Radioscopy of a Handsonic HPD-15 [Blog post]. Retrieved February 21, 2015, from http://francksmith.blogspot.ca/2010/11/inside-drum-synth-radioscopy-of.html

Snyder, J. S., Hannon, E. E., Large, E. W., & Christiansen, M. H. (2006). Synchronization and continuation tapping to complex meters. *Music Perception, 24,* 135–146.

van Vugt, F., & Schultz, B. G. (2015). Taparduino v1.01. *Zenodo*, 16178. doi:10.5281/zenodo.16178