LUCS Minor 17, 2011. ISSN 1104-1609.

Construction of a Low Latency Tapping Board.

Rasmus Bååth

Abstract

This technical report describes the construction of a tapping board to be used in sensorimotor synchronization tasks where the timing of participants' taps are to be registered. The tapping board is designed to be comfortable to use and to register taps with millisecond accuracy.

1 Introduction

Sensorimotor synchronization (SMS) occurs when an agent synchronizes some movement to a predictable external event. Typical examples of SMS are walking to the pace of a drum, dancing or making music in an ensemble. One of the most common paradigms when investigating human SMS is the finger tapping task (Repp, 2005). The finger tapping task was introduced more than a century ago (see Stevens (1886) for an early example) and in its basic form a participant is asked to tap with his or her finger in synchrony with an isochronous sequence of sounds. The timing of the taps is recorded and can be used calculate different measures of timing error, for example the standard deviation of the sound-tap asynchrony. There are many variation of this basic task, the amount of auditory feedback the participant is given can be varied, the participant can tap on a surface or freely flex the finger, the synchronization phase can be followed by a continuation phase where the sound sequence is muted while the participant continues tapping at the same tempo. This last modification is common when trying to dissociate the timing error from the motor error using, for example, the influential model of Wing and Kristofferson (1973).

In order to conduct a finger tapping study one needs an apparatus to play the sound sequence to the participant and to record the timing of participant's taps. If one is interested in the relation between successive taps the recorded timing of the taps need high *reliability*, that is, if the taps are perfectly periodical there should be a low amount of temporal jitter in the recorded timing. If one is interested in the relation between the taps and the sound onsets both the sound playback and the recorded timing needs high reliability and low *latency*. A sound needs to start playback immediately when playback is initiated and there should be no systematic discrepancy between the timing of a tap and the recorded timing. An example of the result of latency and jitter when recording timings in a sequence of taps is given in figure 1.

One apparatus that play sounds and record key presses is a standard personal computer (PC). It would be convenient to use a PC as they are readily available but there are some issues that makes it problematic to use a PC in a tapping study:

- There can be considerable temporal jitter and latency when playing sound through a PC. This of course depends on the brand and setup but a standard PC with a consumer grade sound card running Windows can result in audio delays up ranging from 10 ms to 250 ms depending on the CPU load (MacMillan et al., 2001).
- There can also be temporal jitter and latency in the registration of key presses. No recent article has been found that measures key press latency but Lane and Ashby (1987) estimates it to around 6 ms on a first generation Macintosh computer.

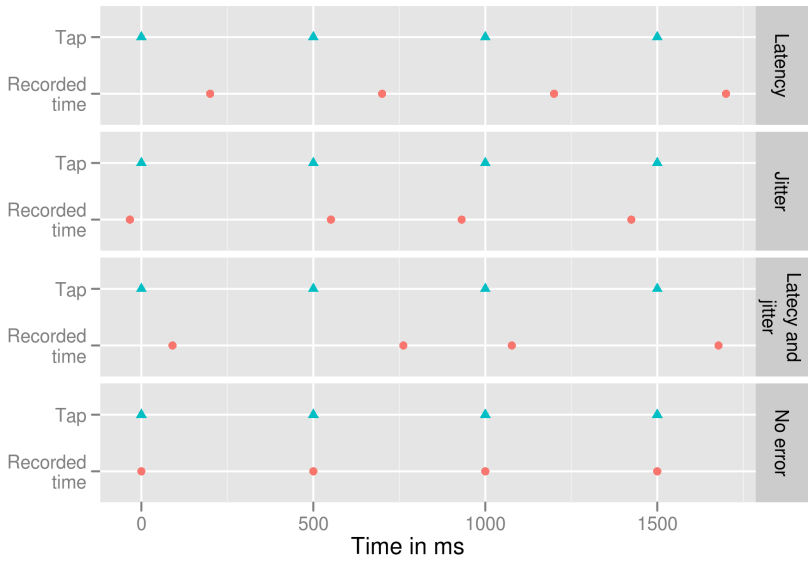

Figure 1: The result of latency and jitter when recording the timing of a sequence of taps.

- It is hard to measure latency and jitter and to separate key press delay from sound delay. Wright et al. (2004) measured the key press to sound delay on computers running Linux and MacOS and and found delays ranging from 10 ms to 80 ms.
- Computer keyboard keys might not be ideal for tapping tasks. It is not enough to just *tap* a keyboard key, it has to be *pressed*, and most computer keyboard keys makes an audible "click" both when pressed and depressed.

One common way of getting around the problem of the tactile feel of keyboard keys and to possibly decrease latency and jitter is to use a MIDI interface for sound playback and/or registering participants' taps. This approach is common in the literature (see e.g. Repp and Doggett, 2007; Madison, 2001) but still suffers from the problem that it is hard to measure delays in sound playback and tap registration. One reason why it is hard to know the delays in PCs and MIDI equipment is that these are complex, non transparent systems where there are many processes running simultaneously and where access to the hardware is hidden behind layers of APIs and abstractions.

Another solution is then to use a system that is simple, that is dedicated to the task of playing sounds and registering taps and where it is possible to guarantee low upper bounds of the delays. Such a system is the *Arduino* (see fig. 2) which is a open-source electronics prototyping platform that includes, among other things, a 16 MHz processor, a USB port and several input and output pins (Mellis and Banzi, 2007). Using the Arduino remedies many of the problems with using a computer. A program implemented on the Arduino runs close to the hardware and there is no operating system that adds unpredictable delays. Because of this, when using an Arduino it is possible to achieve millisecond accuracy when playing sounds and registering taps. Other advantages of the Arduino are that it is affordable (around $30) and that it is relatively easy to hook it up to custom built hardware. The disadvantages with using it is that it requires knowledge of programming (the Arduino uses a subset of the C++ language) and that there is no straight forward way to connect hardware to it, such as a midi instrument, without some knowledge of electronics. Another limitation is that the Arduino only can produce square wave sounds and, while this is enough as a pacing sound for tapping tasks, playing any other sound requires additional hardware.

This technical report describes the construction of an Arduino based tapping board. The tapping board was designed to be comfortable to use and to register taps with

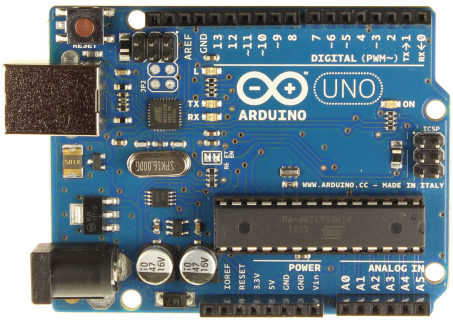

Figure 2: The latest revision of the Arduino prototyping board, the Arduino Uno. The length and width of the Arduino Uno are 6.9 and 5.3 inches respectively. More information can be found on the Arduino home page (www.arduino.cc).

millisecond accuracy. The on-board software was designed to support two types of task: A standard tapping task where a participant synchronizes his or her taps to an isochronous sequence of sounds and a spontaneous motor tempo task where there is no pacing signal and where the participant can tap at any tempo.

2 Construction

The construction consisted of an Arduino, a tapping board with an attached piezo element, a standard 3.5mm stereo jack and a small breadboard that was used to connect the different components. The Arduino was of the "duemilanove" version but this design should work equally well with any Arduino based board. The tapping board consisted of a wooden wrist rest and a 5 cm^2 tapping pad of corrugated fiberboard that rested on a piece of plastic foam of the kind commonly found in foam mattresses. This plastic foam also provided a place to rest for the fingers not involved in the tapping. Below the tapping pad was an attached piezo element that picked up vibrations from the tapping pad. Different materials were tried for the pad but the fiberboard was chosen because it was found to provide a hard surface while still having the elasticity to mediate the taps to the piezo element. The piezo element was connected to an input pin on the Arduino and the stereo jack was connected to an output pin. For a picture of a prototype of the tapping board see figure 4 and for the exact connections made between the components see figure 3.

The Arduino was programmed to handle two types of common tasks, a standard tapping task and a spontaneous motor tempo task (details and source code is found in the appendix). Initiation of the tasks and handling of the resulting data is not made on the Arduino but has to be handled by a PC connected by the USB port.

When the tapping task is initiated the Arduino plays a given number of square wave sounds with a given period and records the timing and amplitude of the taps made on the tapping board. The status of the piezo element is polled more than 10 times every ms. Each sound is associated with a tap that is time stamped at the time of the peak amplitude reading in the time interval with a length of the period of the sequence centered on the sound onset. This method of defining a tap is a robust way of handling noise coming from the piezo element.

In the spontaneous motor tempo task a given number of taps is recorded without there being any pacing sequence. Here a tap is counted as every reading with an amplitude higher than a given threshold. This method of defining a tap is less robust and relies on that the threshold is at carefully adjusted. If it is set to low noise from the piezo element will be counted as taps and if it is set to high real taps will be missed. After a tap there is a 200 ms period in which no tap will be registered; this will limit the tapping rate to

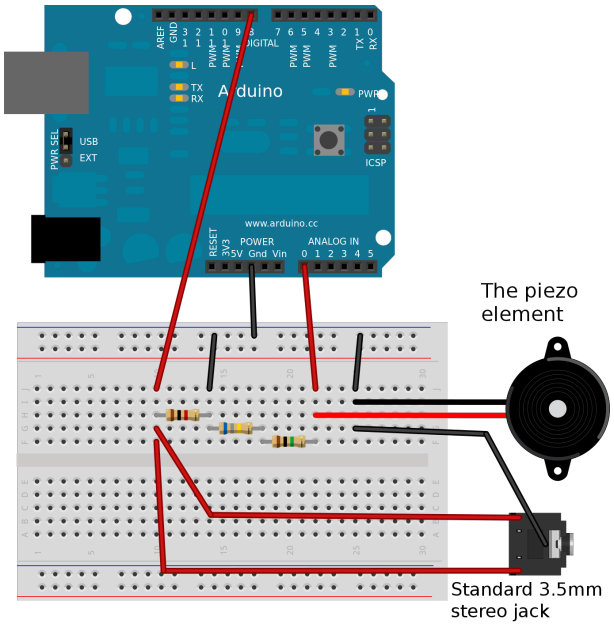

Figure 3: The connections made between the different components. The figure is produced using the Fritzing software (Knörig et al., 2009).

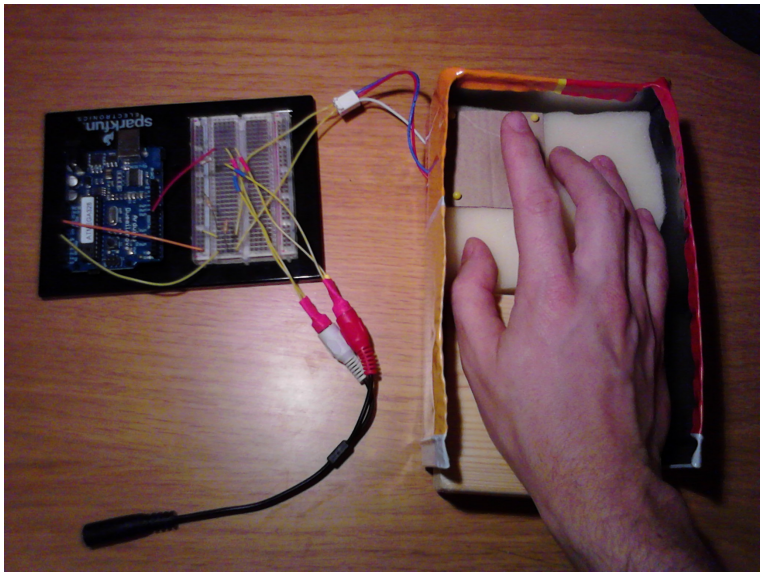

Figure 4: A prototype of the tapping boards with all the components exposed. The piezo element is hidden below the tapping pad.

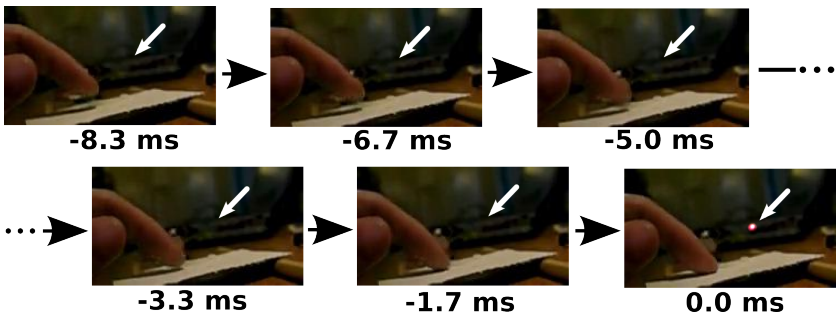

Figure 5: Consecutive frames from a high speed movie of the tapping pad. The white arrow points to a LED that is programmed to light up when the pad is tapped.

five taps per second.

3 Evaluation

Even if the Arduino guarantees a millisecond resolution both when playing sounds and registering taps there is still a need to evaluate the tapping board. First the shape of the response of the piezo element was checked. If there is a large discrepancy between the first amplitude reading above noise level and peak amplitude then the method of registering taps in the tapping task will not work. Testing this, by logging the amplitude with a 0.1 ms resolution, revealed that maximum amplitude was reached within 1-2 ms after the first amplitude reading above noise level.

A high speed camera with an update frequency of 600 Hz (that is, one frame every 1.7 ms) was used to test the total delay of the system (Sanyo Xacti VPC-HD2000). A red LED was connected to the Arduino and programmed to light up as soon as the piezo element registered a tap. This was filmed with the high speed camera and the number of frames between the tap and the lightning of the LED gives an upper limit to the delay of the system. The tap onset and the lightning of the LED always occurred in the same frame (as can be seen in fig. 5) so an upper limit to the delay of the system is 1.7 ms.

References

Knörig, A., Wettach, R., and Cohen, J. (2009). Fritzing: a tool for advancing electronic prototyping for designers. In *Proceedings of the 3rd International Conference on Tangible and Embedded Interaction*, pages 351—-358. ACM.

Lane, D. and Ashby, B. (1987). PsychLib: A library of machine language routines for controlling psychology experiments on the Apple Macintosh computer. *Behavior Research Methods*, 19(2):246–248.

MacMillan, K., Droettboom, M., and Fujinaga, I. (2001). Audio latency measurements of desktop operating systems. In *Proceedings of the International Computer Music Conference*, pages 259–262. sn.

Madison, G. (2001). Variability in isochronous tapping: Higher order dependencies as a function of intertap interval. *Journal of Experimental Psychology: Human Perception and Performance*, 27(2):411–422.

Mellis, D. and Banzi, M. (2007). Arduino: An open electronic prototyping platform. *Conference on Human Factors in Computing Systems.*

Repp, B. and Doggett, R. (2007). Tapping to a very slow beat: A comparison of musicians and nonmusicians. *Music Perception*, 24(4):367–376.

Repp, B. H. (2005). Sensorimotor synchronization: A review of the tapping literature. *Psychonomic Bulletin & Review*, 12(6):969.

Stevens, L. T. (1886). On The Time-Sense. *Mind*, 11(43):393 – 404.

Wing, A. and Kristofferson, A. (1973). Response delays and the timing of discrete motor responses. *Attention, Perception, & Psychophysics*, 14(1):5–12.

Wright, M., Cassidy, R., and Zbyszynski, M. (2004). Audio and gesture latency measurements on Linux and OSX. Technical report, http://cnmat.berkeley.edu/system/files/attachments/latencytest.pdf.

Appendix: Source Code

This is the source code for the program running on the Arduino. It is capable of running standard tapping trials and spontaneous motor tempo trials. The program is not a complete experiment setup but requires a control program running on a host computer. The source code is released under the open source MIT license (http://www.opensource.org/licenses/MIT).

```
// Defining the input and output pins
int piezo_pin = 0;
int speaker_pin = 8;

// Configuration
// The delay between initialisation of the trial and the
// actual start of the trial.
int trial_delay = 500; // ms

int beep_length = 20; //ms
int beep_pitch = 440; // Hz

// The treshold used for the spontaneous motor tempo (SMT)
// task to register a tap
int threshold = 25;

// Minimum possible tap interval in the SMT task
int min_iti = 200; // ms.

// Variable to hold the analog reading from piezo sensor.
int reading;

// Arrays to hold the beep onset times, the tap onset times
// and the amplitude of the taps
unsigned long beep_times[64]; // Max 64 beeps
unsigned long tap_times[64]; // Max 64 taps
int tap_amps[64]; // The amplitudes of the max taps.

void setup(void) {
    Serial.begin(9600);
    pinMode(speaker_pin, OUTPUT);
```

```
    pinMode(piezo_pin, INPUT);
}

void loop(void) {
    int isi;
    int reps;
    // Waits for the input from the serial port that will
    // decide what task is going to be run.
    if (Serial.available() > 0) {
        int inByte = Serial.read();
        // This indicates we should start a tap_trial.
        if(inByte == 't') {
            isi = readInt();
            reps = readInt();
            tap_trial(isi, reps);
            Serial.print("!"); // Marks the end of the trial
        } else if(inByte == 's') { // We start a SMT trial
            reps = readInt();
            smt_trial(reps);
            Serial.println("!"); // Marks the end of the trial
        } else if(inByte == 'b') {
            // We start playing a sequence of test beeps.
            test_beeps();
        } // else do nothing...
    }
}

// Starts a Tap trial with rep number of repetitions at the
// tempo defined by isi.
void tap_trial(int isi, int reps) {
    delay(trial_delay); // So that the trial doesn't start directly.
    unsigned long start_time = millis();
    unsigned long stop_time =
        start_time + long(isi) * long((reps + 1)) - long(10);
    int beep_count = 0;
    unsigned long next_beep = start_time + isi;

    // When we start and stop listening for the tap
    // corresponding to the current beep.
    unsigned long start_listen = next_beep - isi / 2;
    unsigned long stop_listen = next_beep + isi / 2;

    // Variable to hold the maximum amplitude of the registered tap
    int max_amp = -1;
    // Variable to hold the time of the registered tap wich
    // is defined as occuring at the peak amplitude.
    unsigned long peak_time = start_time;

    unsigned long curr_time = millis();
    while(curr_time < stop_time) {
        curr_time = millis();
        if(curr_time > start_listen) {
            reading = analogRead(piezo_pin);
            if(reading > max_amp) {
```

```
                max_amp = reading;
                peak_time = curr_time - start_time;
            }
            if(curr_time > stop_listen) {
                tap_times[beep_count] = peak_time;
                tap_amps[beep_count] = max_amp;
                peak_time = curr_time - start_time;
                max_amp = -1;
                beep_count++;
                start_listen = next_beep - isi / 2;
                stop_listen = next_beep + isi / 2;
            }
        }
        if(curr_time >= next_beep) {
            beep_times[beep_count] = curr_time - start_time;
            tone(speaker_pin, beep_pitch, beep_length);
            next_beep = next_beep + isi;
        }
    }

    // Now printing the result back through the serial port.
    // Format is: beep_onset, tap_onset, tap_amplitude
    for(int i = 0; i < reps; i++) {
        Serial.print(beep_times[i]);
        Serial.print(",");
        Serial.print(tap_times[i]);
        Serial.print(",");
        Serial.println(tap_amps[i]);
        beep_times[i] = 0;
        tap_times[i] = 0;
        tap_amps[i] = 0;
        delay(10);
    }
}

// Runs a spontaneous motor tempo task with
// rep number of repetition.
void smt_trial(int reps) {
    happy_blip();
    int tap_count = 0;
    do { // Wait for first tap...
        reading = analogRead(piezo_pin);
    } while(reading < threshold);

    unsigned long start_time = millis();
    unsigned long curr_time = start_time;

    while(tap_count < reps) {
        if(reading >= threshold) {
            tap_times[tap_count] = millis() - start_time;
            tap_amps[tap_count] = reading;
            tap_count++;
            delay(min_iti);
        }
```

```
        reading = analogRead(piezo_pin);
    }

    // Now printing the result back through the serial port.
    // Format is: tap_times, tap_amplitude
    for(int i = 0; i < reps; i++) {
        Serial.print(tap_times[i]);
        Serial.print(",");
        Serial.println(tap_amps[i]);
        tap_times[i] = 0;
        tap_amps[i] = 0;
        delay(10);
    }
}

// Plays a sequence of beeps
// which is useful for calibrating speakers.
void test_beeps() {
    for(int i = 0; i < 16; i++) {
        delay(600);
        tone(speaker_pin, beep_pitch, beep_length);
    }
}

// Plays a "happy" two note sound
void happy_blip() {
    tone(speaker_pin, 400, 20);
    delay(20);
    tone(speaker_pin, 600, 20);
}

// Reads a two byte integer from the serial port.
int readInt() {
    int i;
    byte b;
    while(Serial.available() == 0) { }
    b = Serial.read();
    i = b;
    while(Serial.available() == 0) { }
    b = Serial.read();
    i |= b << 8;
    return i;
}
```