Tap Arduino: A cross-platform tapping device for sensorimotor experiments and auditory feedback

Reference Manual, Version 1.2, 2015

Benjamin G. Schultz & Floris T. van Vugt

29 June 2015

Contents

1. Introduction

Tap Arduino is a package of C code and Python scripts that can be used for a variety of psychology experiments to record to-the-millisecond responses and present auditory feedback with low latency (down to sub-millisecond; Schultz & van Vugt, 2015). Tap Arduino can be installed on any operating system (OS; Windows, Mac, and Linux) and does not put a heavy load on the CPU. The secret to the success of Tap Arduino is the Arduino microcontroller, an open-source electronics platform that is easy to learn (Arduino, 2015). The Arduino reads signals from a force sensitive resistor (FSR) and produces auditory feedback when the pressure goes above a user-defined threshold. The Arduino is able to timestamp incoming signals with microsecond precision and send timing information to be read by Python via the USB port. As such, the timing provided by the Arduino will be independent of the computer it is running on. The hardware guide we provide presents several options to produce auditory feedback with low latency, depending on how complex the sound is that you want to present.

Tap Arduino software is provided at no charge (but without warranty) under the GNU Public License agreement (GPL v3, 2015).The GNU license allows you to use and modify the software in any way you like for “personal use”, but that any further distribution of the modified software must be done in source code form under the GNU license agreement. Tap Arduino is available for unlimited use and modification, and can be downloaded from our repository: https://zenodo.org/record/16168# (van Vugt & Schultz, 2015).

If you have comments, bug reports, bug fixes, enhancement requests, or coded enhancements, please let us know so we can deal with them and/or choose to incorporate them into the standard version of Tap Arduino. We would be happy to help you with any experiments intend to perform, and please send us any scripts you use in your experiment that you think would help the community. Send comments and questions to benjamin.glenn.schultz@gmail.com or f.t.vanvugt@gmail.com.

Please cite Schultz and van Vugt (2015) in the write-up of any research you do that uses Tap Arduino. We hope you enjoy using Tap Arduino as much as we have enjoyed putting it together!

2. Software Installation

This section explains how to install the software necessary to run Tap Arduino. If you encounter any problems, please let us know the nature of the malfunction and the OS you are using. You will need to download the Tap Arduino package from: https://github.com/florisvanvugt/taparduino

2.1 Arduino Installation

Dedicated and very user-friendly software is available that allows you to upload code to Arduino, called the Arduino IDE (Integrated Development Environment). You can download the Arduino IDE from here: http://arduino.cc/en/Main/Software by selecting your OS, then follow the instructions. This is the software you will use when you upload our C codes (and, eventually, your own) onto the Arduino. Once the Arduino software is installed you will be able to test some of the basic functions of the Arduino outlined in sections 3.1 and 3.2.

In some older versions Linux (e.g., Ubuntu 12.04), the Arduino IDE might be installed incorrectly or install an older version of the Arduino IDE. There are two solutions: 1) upgrade to a more recent version of Ubuntu (e.g., 14.04), or 2) download the executable file. To download the executable file, go the arduino webpage, go to Download, and download the Linux archive (32 bits or 64 bits depending on your system). This will give you a .tar.xz file to download (something like arduino-1.6.1-linux32.tar.xz). Put it somewhere where you can find it, then go there with the file manager, and extract the archive by right-clicking on it and selecting "Extract here". This should create a new folder in the same place, called something like arduino-1.6.1. Inside, there should be an executable file called "arduino". Double click on it, and you should be prompted whether you want to run "arduino' or display its contents: select "Run".

If you are using Linux, the first time you run the Arduino IDE it may ask you to join the dialout group. Click “Add” and restart your computer for these settings to take effect. What this does is ensure that your user account has permissions to read from serial communication ports.

In case you want to learn more about Arduino and developing code for it, please refer to the excellent tutorials on http://arduino.cc/en/Tutorial/HomePage. However, such detailed knowledge is not required if you want to use our existing scripts.

2.1.1 Find your serial communication port

The Arduino device will communicate with your computer through a USB cable. Once connected, your OS will set up a so-called *serial communication port* that connects to Arduino. You need to find out which port this is. In Windows (XP, 7, and 8/8.1) you can check your device manager and look for your Arduino device. It should show "(COM#)" under the device, where "#" is the number of the serial communication port (called COM port in Windows) the device is using. If you are using a Mac or Linux, you can type the following into the terminal: *ls /dev/tty.**

This will produce a list of all serial ports. By comparing this list before and after connecting your USB cable, you can find which serial port newly appeared when Arduino connected, and that will be your designated port. If you are not using other devices, the port that is assigned by default is */dev/ttyACM0*

Optionally, you can instruct Mac or Linux to always map the Arduino to a particular port, by writing an "*udev*" rule. Please refer to an "*udev*" tutorial for this option, since it is beyond the scope of this manual.

2.1.2 Uploading (flashing) code to the Arduino

To record taps and make auditory feedback with the Arduino, you will need to upload the Tap Arduino codes (beginning with "fsr_") to the Arduino. Open up the Arduino IDE. Uploading code to the Arduino is done by opening the code you want to upload, connecting your Arduino through USB, and clicking on the upload icon (a right arrow) or by going to "File" and clicking "Upload". If the code is uploaded correctly, you will receive some white text in the black box at the bottom of the Arduino IDE. If something goes wrong, you will receive some orange text that will give some clues as to what the error is (see section 4.2).

2.1.3 Importing Arduino libraries

If you choose to use the Wave Shield, you will need to download the Wave Shield library (WaveHC) from here: https://code.google.com/p/wavehc/downloads/list and then import the WaveHC library through the Arduino software following the instructions found here: http://arduino.cc/en/guide/libraries. Note that only the folder called "WaveHC" should be imported, not the entire archive.

2.2 Python Installation

Once you have uploaded the code to Arduino using the Arduino IDE and built the device (see section 3) you will be able to record taps and send them through the USB serial communication port. In order to facilitate the reading of this data we supply a simple graphical user interface (GUI) written in Python. This requires a standard Python installation and the pySerial module.

Download Python 2.7 (any sub-version) for your OS from here: https://www.python.org/downloads/. We have not tested these scripts with Python 3.0 and greater, and cannot confirm that it will work. You may want to consider installing the 32-bit version (even if your OS is 64-bit) due to some issues with the registry not always identifying Python modules. Within your Python distribution, you will also need TKinter. This is most likely already included by default. If not, go to https://wiki.python.org/moin/TkInter

In Ubuntu, this software installation step can be done using the following command: *sudo apt-get install python python-serial*

(Note: This step also installs the Python serial module, so you can skip sections 2.2.1 and 2.2.2).

2.2.1 Download the pySerial module

To use the Tap Arduino GUI, you will need to download and install the serial.py module from here: https://pypi.python.org/pypi/pyserial

For a Mac or Linux, you will need to download the archive (ending in ".tar.gz"). For Windows, you can install the executable ("pyserial-2.7.win32.exe") file or download the archive. If you downloaded the archive, you should extract this into your Python27 folder (if you use Ubuntu, and you used the apt-get command in section 2.2 above, you don't need to do this and can skip to section 2.2.4).

2.2.2 Installing the pySerial module

2.2.2.1 Mac and Linux

To install the pySerial Python module in Mac or Linux, you will need to extract the archive to a temporary folder. Open the terminal and navigate to the directory where you extracted the pySerial archive (e.g., "/pyserial-2.7/") by typing:

cd Desktop/pyserial-2.7/

Then you can install the module by entering the following into the terminal:

sudo python setup.py install

2.2.2.2 Windows command line

If you use a Windows device and you have downloaded the archive, you can open the command line by searching for “cmd.exe” from the start menu and opening it. You would then type the following (assuming you installed Python in your C: directory, and extracted pyserial into the Python27 folder). First we set the python path (this only needs to be done once after installing python):

set path=%path%;C:\Python27

Then we install the Py Serial module:

python Python27\pyserial-2.7\setup.py install

2.2.2.3 Windows executable file

If you use Windows and prefer to use the executable file, then you can select and open the file “pyserial-2.7.win32.exe”. Follow the prompts to install the python module. Note that we have experienced some difficulties with modules being identified in the registry when using 64-bit Windows operating systems. If you experience this problem and have no idea what the registry is or how to change it, then we suggest you use the command line installation described above (see section 2.2.2.2). If you know what you are doing and want to edit the registry, there are some forums with instructions on how to do so. We remind the reader that they should always back up their registry before editing. The authors do not accept any responsibility for any damages or losses that result from the use of the Tap Arduino package, Arduino IDE, or Python.

If you would like to learn more about downloading Python modules, please visit the Python documentation site: https://docs.python.org/2/install/

2.2.3 Installing other Python modules

You will probably need to install some other Python modules if you would like to run experiments that will be contributed to our repository, but these will be explicitly

stated in the codes as they are uploaded. For the Tap Arduino GUI, only pyserial is required.

2.2.4 Using the Tap Arduino General User Interface

These instructions should help you to get acquainted with the Tap Arduino GUI. Since we have not set up the Arduino hardware yet, upload the code "fsr_silent_cont.ino" to the Arduino (see section 2.1.1). This script will output the Arduino time and a reading of zero (0) from the FSR because it is not yet connected.

1. Run the script capture-gui.py with Python. For a Mac or Linux (Ubuntu) computer, the easiest is to use a terminal, go to the folder where you have downloaded the GUI (e.g., "taparduino-master\gui\"), and type:

   python capture-gui.py

   For Windows, you can simply double-click the "capture-gui.py" file.

2. Enter the correct port for Arduino communications in the text box marked "serial port". See section 2.1.1 above. This will be something like *COM30* on Windows or */dev/ttyACM0* on Mac/Linux.
3. Select a file to write the captured data to. To do this, click on the button "Select". Make sure you don't overwrite an existing file.
4. Select the correct Arduino setting: discrete or continuous. This should correspond to the way you have programmed Arduino (i.e., the script that you have uploaded to Arduino using the Arduino IDE). The Arduino codes for the discrete data type end in "disc" and Arduino codes for the continuous data type end in "cont". In the discrete case, the Arduino will send one data packet for each tap. In the continuous case, the Arduino will communicate the complete recording of the FSR at each point in time (so that you can detect the tap onsets yourself afterwards). For this example, select the continuous data type.
5. Press the green "Capture" button. You should see two columns of data. The first is the time recorded by the Arduino and the second will eventually be the reading of the FSR (when connected).
6. When you are done, press "Stop". All data will be automatically written to a text file and saved as per your preset save filename.

3. Hardware Configuration

Here we give you step by step instructions for how to build the Arduino, connect the sensors and audio outputs, and testing the equipment to make sure everything works as we go along. This should help with trouble shooting later. As a general rule, the colouring of our wires will consistently follow this convention: red is connected to power pins (3.3V or 5V), black is connected to ground pins (GND), green is

connected to analogue inputs (A0 to A5), and purple is connected to audio outputs (pulse-width modulation pins 9, 10, or 11).

3.1 Force Sensitive Resistor (FSR)

This section describes how to connect the force sensitive resistor (FSR) to the Arduino and record responses made using the Tap Arduino GUI. If you do not require auditory feedback in your experiments, then this section should prepare you to start your Tap Arduino experiment(s).

3.1.1 What to buy

For the FSR setup, you will need to purchase an Arduino UNO, an FSR (we use a square one in this example), a breadboard, some hook up wire, a 2-pin 3.5mm screw terminal (optional), and a 10kΩ resistor with the colours brown, black, orange, and gold (see Appendix Y). The FSR is quite fragile, and there are alternatives to soldering wires to the FSR (see https://learn.adafruit.com/force-sensitive-resistor-fsr/connecting-to-an-fsr). If you do not have much (or any) experience with soldering, we recommend you use screw terminals or clamp connectors (i.e., alligator clips).

3.1.2 Setup and schematics

While the Arduino is OFF (the power and USB are not connected), arrange the cables as follows (see Figure 1):

1. *Carefully(!)* solder the end of the red hook up wire to one side of the FSR (or screw terminal).
2. *Carefully(!)* solder the end of the green hook up wire to the other side of the FSR (or screw terminal).
3. (Optional step) If using a screw terminal, insert the two FSR pins into the two pin holders and use a small screwdriver to screw the pins in place.
4. Connect the other end of the red wire into the Arduino's 5V pin, and connect the other end of the green wire into the breadboard.
5. Connect a second green wire from the same row of the breadboard as the first green wire to the Analogue 0 pin (A0) of the Arduino.
6. We use the 10kΩ resistor as a voltage divider between the ground pin and the incoming signal from the FSR. Connect one end of the resistor to the same breadboard row as the green wires, and the other end to a different breadboard row.
7. Connect one end of the black wire to the same breadboard row as the resistor (but NOT the same row as the green wires), and the other end of the black wire to the Arduino's ground (GND) pin.

Congratulations! You have just connected the FSR to the Arduino. Now we should test that everything works.

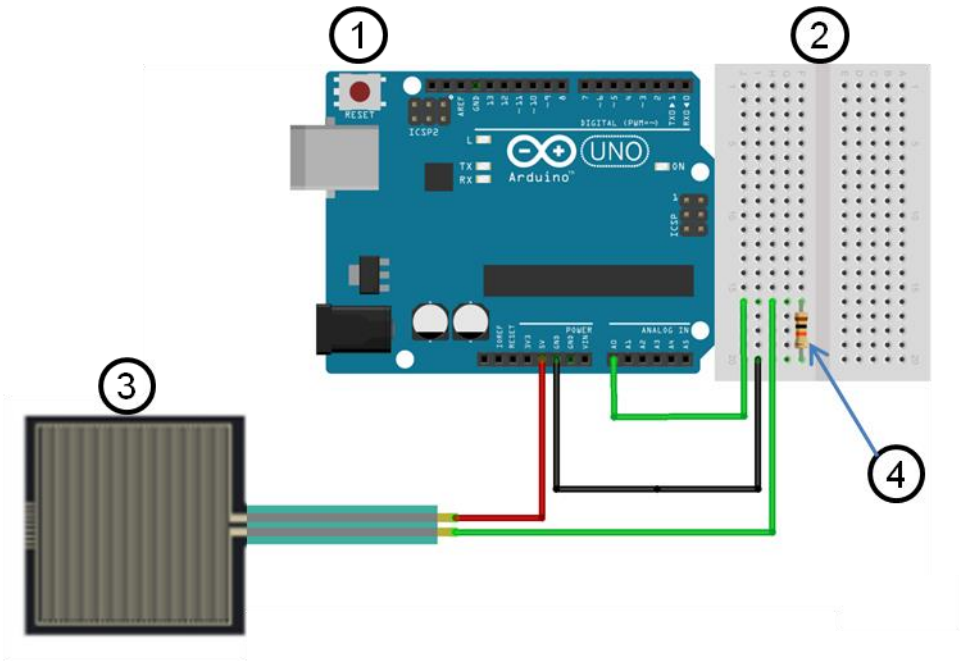

Figure 1. Schematic of the wiring for the force sensitive resistor (FSR) and the Arduino. Labels refer to the following: 1. Arduino UNO, 2. Breadboard, 3. FSR, 4. 10kΩ resistor. See text for construction instructions. Figure produced using fritzing (Knörig, Wettach, & Cohen, 2009).

3.1.3 Testing the FSR

Once you have connected the FSR to the Arduino, we can test that the FSR is working. Plug the USB cable into the Arduino (with the other end connected to your computer). The lights on the Arduino should come on, indicating that you are now connected and the power is on (the Arduino is USB powered). If the lights do not come on, then your USB cable might need replacing or the Arduino you purchased may have shuffled off this mortal coil (i.e., died) and you will need to speak to your retailer. If the lights go on, then we may proceed.

Open the “fsr_silent_cont.ino” sketch using the Arduino software. Upload the sketch as shown in section 2.1.2 and open the Tap Arduino GUI as shown in section 2.2.3, making sure you select “continuous” as the “data type”. You should see something

like the output in Figure 2. Try tapping on the FSR and see what it looks like. This is how continuous data is read into the Arduino from the FSR. You can also test the discrete data capture that sends the onset, offset, and maximum force for each tap using the "fsr_silent_disc.ino" code on your Arduino, and selecting "discrete" as your "data type". If everything is working properly, then you should receive a new line of data for every tap. If you are not interested in using auditory feedback, then this is all you need to start collecting data for your next experiment!

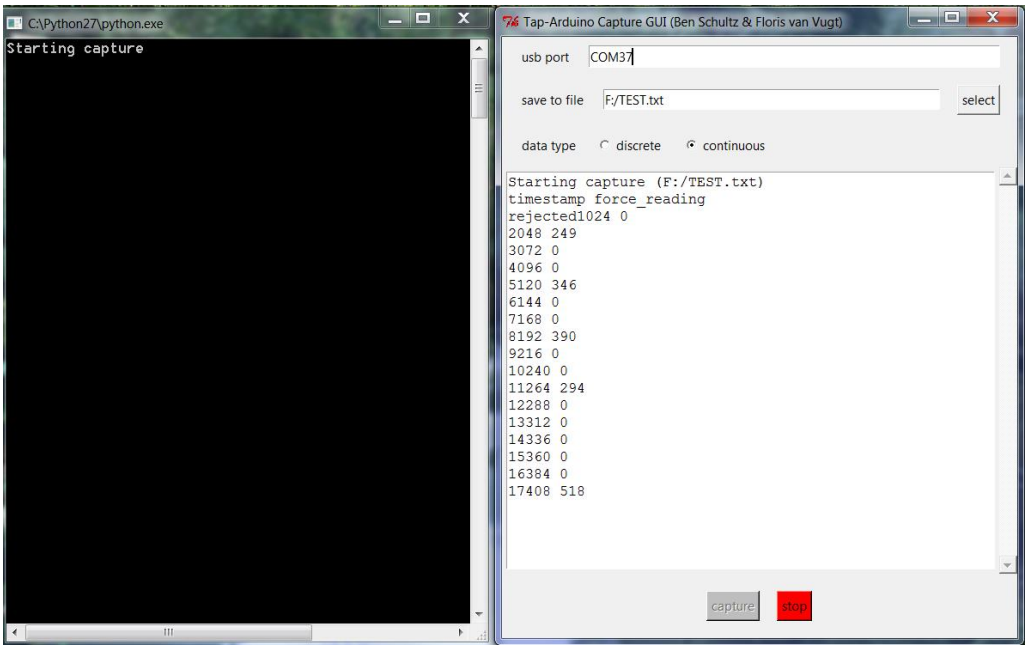

Figure 2. Example of output for testing the FSR (using "fsr_silent_cont.ino").

3.2 Pulse-width modulation tones

The most affordable method to create auditory feedback is to send a cycling waveform from the pulse-width modulation (PWM) pin to a headphone jack. This method also produces sub-millisecond tap-to-feedback latencies (Schultz & van Vugt, 2015). This section describes how to connect the PWM to the headphone jack to produced auditory feedback when the FSR is triggered, and record responses made using the Tap Arduino GUI. We assume that you have already connected the FSR as described in section 3.1.

3.2.1 What to buy

For the PWM setup, you will need to purchase an Arduino UNO, an FSR (we use a square one in this example), a headphone jack (we use a 3.5mm TRRS jack breakout board, see Figure 3), male headers (4 pins), a breadboard, some hook up wire, and two 10kΩ resistors with the colours brown, black, orange, and gold (see

Appendix A). We chose the TRRS jack with the breakout board because it is easier to connect the headphone to the breadboard, and to know what the connections refer to (i.e., tip, ring 1, ring 2, and sleeve; see Figure 3): https://www.sparkfun.com/products/11570

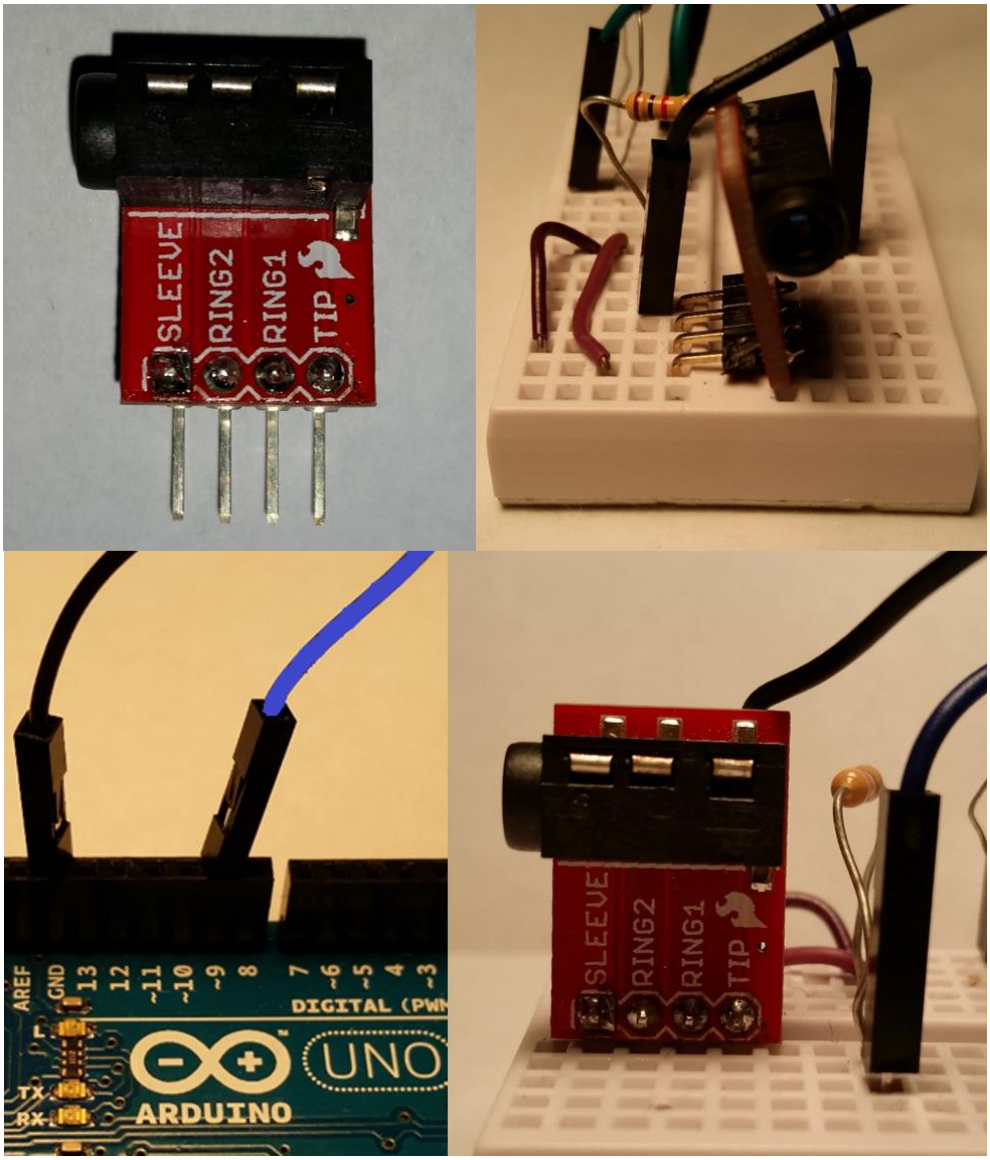

Figure 3. Photos of the headphone jack and breakout board (top panels and bottom right panel) that connects to the PWM pin (~9 pin) and ground (GND pin) shown in the bottom left panel. See Figure 4 for instructions.

3.2.2 Setup and schematics

After the FSR has been connected as described in section 3.1, the following additions can be made. While the Arduino is OFF (the power and USB are not connected), arrange the cables as follows (see Figure 4):

1. Solder the short side of the male header pins to the four pins of the headphone breakout board (labelled tip, ring 1, ring 2, and sleeve). Once cooled, insert the long end of the male header pins into the breadboard, ensuring that each pin is in a different row.
2. Connect one end of a purple wire to the Arduino's digital pin 9 (~9), and the other end to an empty row of the breadboard.
3. Connect one end of the 10kΩ resistor to the same breadboard row as the purple wire, and the other end to a different empty breadboard row.
4. Connect two purple wires to the end of the 10kΩ resistor and the other ends of the purple wires connect to the breadboard row containing the tip and ring 1 of the headphone jack (or breakout board; some soldering may be required). The tip and ring 1 of the headphone jack connect to the left and right speaker of your output device.
5. Connect one end of the black wire to the row containing ring 2 and the other end to a ground (GND) pin of the Arduino.

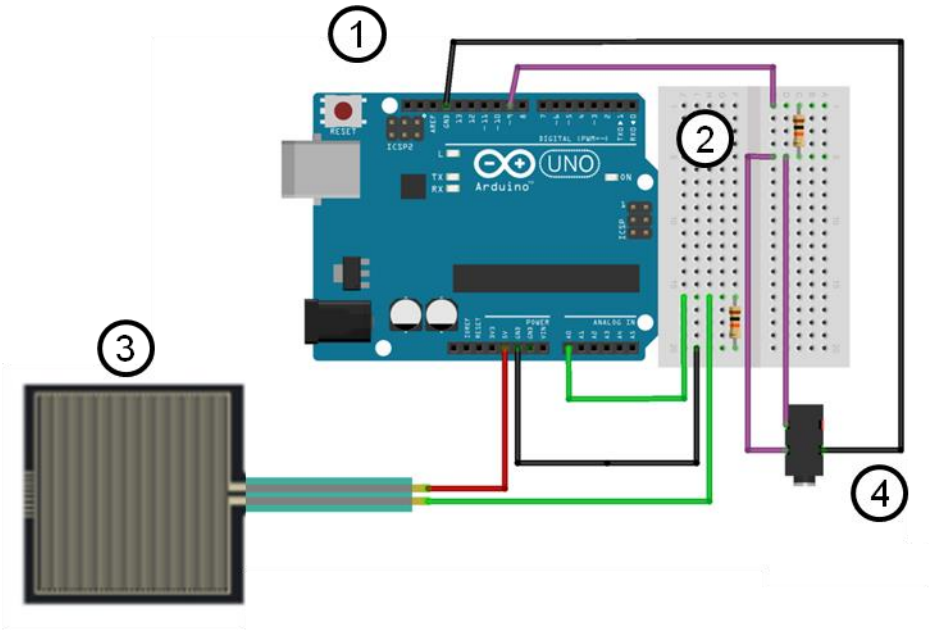

Figure 4. Schematic of the wiring for the force sensitive resistor (FSR) and the PWM tone through a headphone jack. Labels refer to the following: 1. Arduino UNO, 2. Breadboard, 3. FSR, 4. A TRRS 3.5mm headphone jack. Both resistors are 10kΩ (brown, black, orange, gold). See text for construction instructions. Figure produced using fritzing (Knörig, Wettach, & Cohen, 2009).

3.2.3 Testing auditory feedback

To test auditory feedback, connect any speaker or headphone set to the headphone jack. Now upload the "fsr_tone_cont.ino" or "fsr_tone_disc.ino" onto the Arduino as shown in section 2.1.2. When you tap, you should hear a single tone of your specified wave type (sine, square, saw), frequency (variable "tone_freq"), and

duration (variable “tone_dur”). These values can be changed in the “fsr_tone_cont.ino” and “fsr_tone_disc.ino” Arduino scripts. You may also want to check that the responses are being recorded by the Tap Arduino GUI, as shown in section 3.1.3.

3.3 Wave Shield

While the PWM setup is useful for presenting simple waveform feedback, experimenters sometimes like to present more complex stimuli to participants. The Wave Shield (Adafruit, 2015) is an Arduino compatible method for presenting participants auditory feedback corresponding to any .wav file that can fit onto a secure disk (SD) card. Tap-to-feedback latencies were approximately 2.5ms for the Wave Shield (Schultz & van Vugt, 2015), so still well-below the perceivable threshold, or the threshold that affects behaviour (see Aschersleben & Prinz, 1997). This section describes how to connect the Wave Shield and FSR to the Arduino to produce auditory feedback when the FSR is triggered, and record responses made using the Tap Arduino GUI. Note that the Wave Shield can produce quite loud auditory feedback, so please test the Wave Shield audio with your headphones off or with the volume low to prevent hearing damage or deafness.

3.3.1 What to buy

For the Wave Shield setup, you will need to purchase an Arduino UNO, an FSR (we use a square one in this example), a breadboard, some hook up wire, a 2-pin 3.5mm screw terminal (optional), a Wave Shield, an Arduino Stackable Header Kit, and a 10kΩ resistor with the colours brown, black, orange, and gold (see Appendix A).

3.3.2 Setup and schematics

The fine folks at Adafruit have provided a terrific guide for building the Wave Shield here: https://learn.adafruit.com/adafruit-wave-shield-audio-shield-for-arduino. There is one small divergence, when you reach the section that says:

> *Next, break the 36-pin header strip into smaller sections so that the shield can be placed on the Arduino. You can use pliers or diagonal cutters. Clip off 2-6pin and 2-8pin pieces.*

Instead of header strips, we’re going to use the stackable Arduino headers. These can be soldered using the guide here: https://learn.adafruit.com/adafruit-proto-shield-arduino/solder-it. Once you have built the Wave Shield and connected it to the Arduino, you will need to connect the FSR as done in section 3.1.2, except you will connect the FSR to the Wave Shield pins of the same names.

3.3.3 Configuring the Secure Disk (SD) card and sounds

The method for formatting the SD card is outlined on the Adafruit webpage: https://learn.adafruit.com/adafruit-wave-shield-audio-shield-for-arduino/sd-card. Once the SD card has been formatted, you will need to add the sound files to the SD card. We provide the example sound “SQ.wav”, a 1046.5Hz square wave of 20ms duration. If you choose to use your own sounds, you will need to reformat the sounds like so: https://learn.adafruit.com/adafruit-wave-shield-audio-shield-for-arduino/convert-files. You can change the sound used as auditory feedback by the Wave Shield by changing the variable “wavFile” in “fsr_wave_cont.ino” or “fsr_wave_disc.ino”.

3.3.4 Testing auditory feedback

To test auditory feedback of the Wave Shield, connect any speaker or headphone set to the headphone jack on the Wave Shield. Before uploading the Arduino scripts, you will need to download and import the “WaveHC” library. You can download the “WaveHC” library from here: https://code.google.com/p/wavehc/downloads/list. Then you will need to import the “WaveHC” library using the Arduino software, following the instructions in section 2.1.3. Before uploading the code, remove your headphones or turn the volume down using the potentiometer (the black wheel) to prevent damage. Now upload the “fsr_wave_cont.ino” or “fsr_wave_disc.ino” onto the Arduino as shown in section 2.1.2. When you tap, you should hear the sound specified by the variable “wavFile”. You may also want to check that the responses are being recorded by the Tap Arduino GUI, as shown in section 3.1.3.

4. Troubleshooting

Anything that can go wrong, will go wrong (*Murphy's Law*). We know from experience that things won't always go smoothly. Here we have tried to pre-empt issues that may arise and the quickest way to finding the problem and fixing it.

4.1 Problems with the Tap Arduino GUI

We haven't had any problems with the Tap Arduino GUI yet, but some issues are bound to come up at some point. Here are the usual suspects and solutions for problems you may encounter with the interface.

1. Is it plugged in? Are the Arduino lights on? If the Arduino lights are not on, then there is either something wrong with the USB cable or the Arduino might be broken (or "bricked"). These two problems can only be solved by replacing the cable or Arduino.
2. Is the USB being recognized? If the Arduino USB is connected through a secondary source (e.g., a keyboard or a monitor) then it might not be read correctly by the computer as a serial port. Ensure that the USB is directly connected to a computer USB port. This issue will affect the uploading of Arduino code and the reading of data through the USB.
3. Is the wiring correct? Double-check (and triple-check!) that the wires are in the correct pins, that the wires are in the correct rows, and that no wires are connected that shouldn't be. See sections 3.1.2 and 3.2.2 for examples of the wiring and connections.
4. Do you have the correct serial communication port number? It is possible that you have selected the wrong serial communication (COM) port number. Check again by following the instructions in section 2.1.1.
5. Have you selected the correct data type in the Tap Arduino GUI? For our Arduino scripts, any code that ends in "cont" outputs continuous data and any code that ends in "disc" outputs discrete data.
6. Did the Arduino code upload correctly? Follow the instructions in section 2.1.2. Check the black display in the Arduino GUI when you upload your script. If there is any orange text, then something went wrong. We suggest that you go back to the source code to make sure that no errors have been introduced. If you are still receiving errors (i.e., orange text), ensure that you have the correct libraries imported and that your variables have been defined.
7. If you have gone through all of these steps and you are still having issues, please email us: benjamin.glenn.schultz@gmail.com or f.t.vanvugt@gmail.com.

4.2 Using the serial monitor

The Arduino serial monitor can also be used to test the output of the Arduino if you are having difficulties with the Tap Arduino GUI. The serial monitor can be accessed in the Arduino IDE by going to the "Tools" tab and clicking on "Serial Monitor". We have included parts of our script that can be read in the Arduino serial monitor, just in case you are having difficulties with Python. These sections are in the "collectData()" function in the functions section of the Arduino code, preceded by "Send data to the serial port (for debugging in Arduino serial manager)". If you remove the "//" at the beginning of the five lines following this sentence, and add "//" to the beginning of the five lines following "Send data to the serial port", then you can use the serial monitor to read the output from the Arduino. The serial monitor is also useful for receiving errors that might arise while using the Wave Shield (see section 4.3). Note that the baudrate in the serial monitor (in the bottom right corner of the Arduino IDE) has to match the baudrate used in the Arduino code. For the discrete Arduino codes (*disc.ino) the baudrate must be set to 9,600, and for the continuous Arduino codes (*cont.ino) the baudrate must be set to 115,200.

4.3 Auditory feedback problems

As a general rule, if you are experiencing any problems with the auditory feedback, you should check the following:

1. Is it plugged in? Are the Arduino lights on? If the Arduino lights are not on, then there is either something wrong with the USB cable or the Arduino might be broken (or "bricked"). These two problems can only be solved by replacing the cable or Arduino.
2. Is the wiring correct? Double-check (and triple-check!) that the wires are in the correct pins, that the wires are in the correct rows, and that no wires are connected that shouldn't be. See sections 3.1.2 and 3.2.2 for examples of the wiring and connections.
3. Is the Tap Arduino GUI still receiving data? If Tap Arduino is still recording data (particularly discrete data), then it is likely that something is wrong with the PWM wiring or the Wave Shield. Alternatively, there might be something wrong with your audio output device (e.g., headphone set or speakers).
4. Is the FSR data noisy for continuous data? If Tap Arduino records sporadic or noisy data using the continuous data capture, then it is likely that the FSR is broken. This can happen if the FSR is bent, exposed to high temperatures, or if the pin strip is ripped. If this is the case, try another FSR. If another FSR works, then the other FSR is probably damaged and unusable for precise data recording.
5. Did the Arduino code upload correctly? Check the black display in the Arduino GUI when you upload your script. If there is any orange text, then something went wrong. We suggest that you go back to the source code to make sure

that no errors have been introduced. If you are still receiving errors (i.e., orange text), ensure that you have the correct libraries imported and that your variables have been defined.

6. Is your audio output device connected correctly? Ensure that the speaker or headphone jack is securely (and completely) inside of the input of the TRRS jack or Wave Shield.
7. Does your audio output device work with other devices? Check to see if your speakers or headphone set work with other devices. If not, then there might be something wrong with your audio output device.
8. If you have gone through all of these steps and you are still having issues, please email us: benjamin.glenn.schultz@gmail.com or f.t.vanvugt@gmail.com.

4.3.1 I'm still not getting any sound!

If you are using the PWM method, check that your selected frequency is perceivable to the human ear (usually between 20Hz and 20,000Hz) and that the duration is long enough to be heard (at least 5ms). If you are using the Wave Shield, check the serial monitor (see section 2.1.4) to see if there is an issue with reading the sound file (.wav), or if the sound file cannot be found. The Arduino code is case sensitive, so check the filename and ensure that the file has been uploaded onto the SD card with the correct formatting (see section 3.3.3). The Wave Shield also has a volume control wheel, so check to see if you have the volume turned up enough.

4.2.2 The sound keeps repeating!

Repeating sounds is usually a problem with the signal from the FSR. Check that the FSR is not bent or ripped, and that the user is not touching any of the metal components or wiring. Check the wiring and connections, as this issue can arise from having ungrounded connections.

4.4 I'm having a problem you haven't thought of!

Inconceivable! If you do, however, come across a problem or issue that we have not documented here, please let us know. We would gladly try our best to get you up and running. Please email us: benjamin.glenn.schultz@gmail.com or f.t.vanvugt@gmail.com, with your name, OS, Arduino version, Python version, and a description of your problem.

5. Citing this software

If you use any of the scripts or codes that have been shown in this manual or from the repository (van Vugt & Schultz, 2015), please cite the associated article:

Schultz, B. G. & van Vugt, F. T. (2015). Using an Arduino microcontroller to deliver low-latency auditory feedback in sensorimotor tapping experiments. *Behavior Research Methods.*

5.1 I can't find your reference online!

Just a warning – this paper has not been published (yet) and is, in fact, still being reviewed. In the meantime, we would be happy to send you a draft of the paper until it is accepted.

5.2 Contributions

This project was a joint effort between Benjamin Schultz and Floris van Vugt. Floris van Vugt was responsible the initial Arduino code that sends continuous data to the serial port, the wave table sounds using PWM, the basic communication protocol between the Arduino and Python, and also the Python GUI for data capture. Benjamin Schultz was responsible for the tap-triggered auditory feedback code (for the PWM and Wave Shield), the adaptations of the Wave Shield code, the catches to prevent double taps, and the adapted communication protocol for sending discrete tap data (onsets, offsets, and maximum force).

6. References

Adafruit (2015). Adafruit Wave Shield for Arduino kit, http://www.adafruit.com/product/94

Arduino (2015). http://www.arduino.cc/

Aschersleben, G. & Prinz, W. (1997). Delayed auditory feedback in synchronization. *Journal of Motor Behavior, 29 (1),* 35-46.

GPL v3 (2015), http://www.gnu.org/copyleft/gpl.html

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

Schultz, B. G. & van Vugt, F. T. (2015; Under review). Tap Arduino: An Arduino microcontroller for low-latency auditory feedback in sensorimotor tapping experiments. *Behavior Research Methods.*

van Vugt, F. T. & Schultz, B. G. (2015). Floris T. van Vugt & Benjamin G. Schultz (2015). Taparduino v1.0. *Zenodo. DOI:* 10.5281/zenodo.16168

Appendix A:
List of components for Arduino

Item	Links
Arduino UNO (Revision 3)	http://store.arduino.cc/
Square force sensitive resistor (FSR 406)	http://www.interlinkelectronics.com/standard-products.php
TRRS 3.5mm Headphone Jack Breakout	https://www.sparkfun.com/products/11570
10kΩ Resistor (10kOhm Resistor; brown, black, orange, gold)	https://learn.sparkfun.com/tutorials/resistors/types-of-resistors
Wave Shield v1.1	http://www.adafruit.com/product/94
Mini Breadboard	https://learn.sparkfun.com/tutorials/how-to-use-a-breadboard
USB 2.0 Type A to Type B (NOT the mini USB connector)	https://www.sparkfun.com/products/512
Secure Disk (SD) card	http://www.adafruit.com/products/1294
Hook up wire (kit) Wire colour unimportant	https://www.sparkfun.com/products/124
Screw terminal (3.5mm, 2-pin)	https://www.sparkfun.com/products/8084
Break away headers (Male)	https://www.sparkfun.com/products/116
Arduino Stackable Header Kit	http://www.robotshop.com/en/arduino-stackable-header-kit.html

Appendix B:
Electronics Vendors

Name	Country/Countries	Link
DigiKey	Worldwide	http://www.digikey.com/
Robotshop	U.S.A./Canada/UK/Europe	http://www.robotshop.com
Sparkfun	U.S.A./Canada/UK/Europe/Australia	https://www.sparkfun.com/
JayCar Electronics	Australia	http://www.jaycar.com.au/

Please help us add to this list for your country! Send your country and vendor name and website to benjamin.glenn.schultz@gmail.com or f.t.vanvugt@gmail.com.

Appendix C:
Serial Communication Protocols

The Arduino scripts that we provide here use a simple communication protocol to send data to a computer through the USB cable. This protocol exists in two flavours.

The fsr_*_disc scripts send one data packet per tap. This packet has the following form and is sent in binary fashion.

Byte	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Content	**“B”**	**onset time**		**offset time**		**maximum force**		**“E”**
Encoding	char	int (16 bits)		int (16 bits)		int (16 bits)		char
Description	Marker for the beginning of the packet	The time the tap started (i.e. the time the FSR reading first exceeded threshold).		The time the tap stopped (finger released contact with the tapping surface).		The maximal tapping force readout during this tapping cycle (between tap onset time and tap offset time).		Marker for the end of the packet

The fsr_*_cont scripts send data continuously, containing the raw readings of the FSR. This packet has the following form and is sent in binary fashion.

Byte	**1**	**2**	**3**	**4**	**5**	**6**
Content	**“B”**	**timestamp**		**FSR reading**		**“E”**
Encoding	char	int (16 bits)		int (16 bits)		char
Description	Marker for the beginning of the packet	The timestamp (according to the Arduino internal clock) of the time at which the measurement was made.		The raw reading of the FSR sensor (range 0-1024).		Marker for the end of the packet