

**The Schultz Cigarette Burn Toolbox for measuring dynamic video framerates  
and synchronizing video stimuli with neural and behavioral responses**

**Reference Manual, Version 1.0, 2019**

**Benjamin G. Schultz**

**15 May 2019**

## Contents

1. Introduction .....	4
2. Software Installation .....	6
2.1 MATLAB Installation .....	6
2.2 Arduino Installation .....	6
2.2.1 Find your serial communication port .....	7
2.2.2 Uploading (flashing) code to the Arduino .....	7
2.2.3 Arduino Serial Monitor .....	8
2.3 Python Installation .....	8
2.3.1 Download the pySerial module .....	9
2.3.2 Installing the pySerial module .....	9
2.3.2.1 Mac and Linux .....	9
2.3.2.2 Windows command line .....	9
2.3.2.3 Windows executable file .....	10
2.3.3 Installing other Python modules .....	10
3. Hardware Configuration .....	11
3.1 Photodiodes .....	11
3.1.1 What to buy .....	11
3.1.2 Setup and schematics .....	11
3.1.3 Testing the Photodiode .....	12
3.1.3.1 Testing the Photodiode with an analog input box .....	12
3.1.3.2 Testing the Photodiode with the Arduino .....	13
4. MATLAB Code .....	15
4.1 MATLAB Code to make flicker videos .....	15
4.2 MATLAB Code to add visual markers .....	15
4.3 MATLAB Code to extract onsets/offsets .....	15
5. Troubleshooting .....	17
5.1 Problems with the Arduino GUI .....	17
5.2 Using the serial monitor .....	18
5.3 The Python Scripts Aren't Working! .....	18
5.4 The MATLAB Scripts Aren't Working! .....	18
5.5 I'm having a problem you haven't thought of! .....	18
6. Citing this software .....	19

6.1 I can't find your reference online! .....	19
6.2 Contributions .....	19
7. References .....	20
Appendix A: List of components for Arduino .....	21
Appendix B: Electronics Vendors .....	22

## 1. Introduction

The Schultz Cigarette Burn Toolbox (SCiBuT) is a package of C code, Python code, and MATLAB scripts that can be used to test temporal latencies that occur when presenting visual stimuli including videos and dynamically presented images. Specifically, the SCiBuT can measure:

1. Average framerates
2. Framerate consistency
3. Missed frames
4. Video and image onset times

The SCiBuT can be installed on any operating system (OS; Windows, Mac, and Linux) and does not put a heavy load on the CPU. In the absence of an analog input box (for EEG setups), the SCiBuT can use an Arduino microcontroller, an open-source electronics platform that is easy to learn (Arduino, 2015). The Arduino can read incoming analog signals from multiple devices and timestamp them to the nearest millisecond. These signals can be simultaneously recorded to show the latency between consecutive visual stimulus onsets and offsets, and between the onset of a visual stimulus and a response with a temporal resolution of 1ms. When using an analog input box (or similar), the temporal resolution of the SCiBuT is only limited by the sampling rate of the device used to record the sensor data (which responds with sub-microsecond accuracy). Such devices include analog input boxes, oscilloscopes, and audio sound cards with at least two audio inputs.

The SCiBuT software is provided at no charge (but without warranty) under the GNU Public License agreement (GPL v3, 2018). 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. The SCiBuT is available for unlimited use and modification, and can be downloaded from the repository:

<https://band-lab.com/scibut> (Schultz, Biau, & Kotz, 2019).

If you have comments, bug reports, bug fixes, enhancement requests, or have coded enhancements yourself, please let me know so I can deal with them and/or choose to incorporate them into the standard version of the SCiBuT. I would be happy to help you with any experiments you intend to perform, and please send me any scripts you use in your experiment that you think would help the community. Send comments and questions to [benjamin.glenn.schultz@gmail.com](mailto:benjamin.glenn.schultz@gmail.com).

Please cite Schultz et al. (2019) in the write-up of any research you do that uses the SCiBuT. We hope you enjoy using the toolbox as much as we have enjoyed putting it together!

## 2. Software Installation

This section explains how to install the software necessary to run the SCiBuT. If you encounter any problems, please let me know the nature of the malfunction and the OS you are using. You will need to download the SCiBuT from:

<https://band-lab.com/scibut>

### 2.1 MATLAB Installation

Install MATLAB following the instructions provided by MathWorks:

<https://nl.mathworks.com/help/install/ug/install-mathworks-software.html>

There are two ways to install the MATLAB scripts in the SCiBuT. You can simply have a copy of the scripts in the folder where you run the scripts, or you can add the folder “SCiBuT” to the path using “Set Path” in MATLAB:

[https://nl.mathworks.com/help/matlab/matlab\\_env/add-remove-or-reorder-folders-on-the-search-path.html](https://nl.mathworks.com/help/matlab/matlab_env/add-remove-or-reorder-folders-on-the-search-path.html)

### 2.2 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 following the instructions. This is the software you will use when you upload the Arduino 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 of 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 later), or 2) download the executable file. To download the executable file, go the Arduino webpage ([www.Arduino.cc](http://www.Arduino.cc)), 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 so your computer can communicate with the Arduino.

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 the SCiBuT scripts.

### 2.2.1 Find your serial communication port

The Arduino device will communicate with your computer through a USB cable. Once connected, your operating system (OS; e.g., Windows, Linux, Mac) 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`

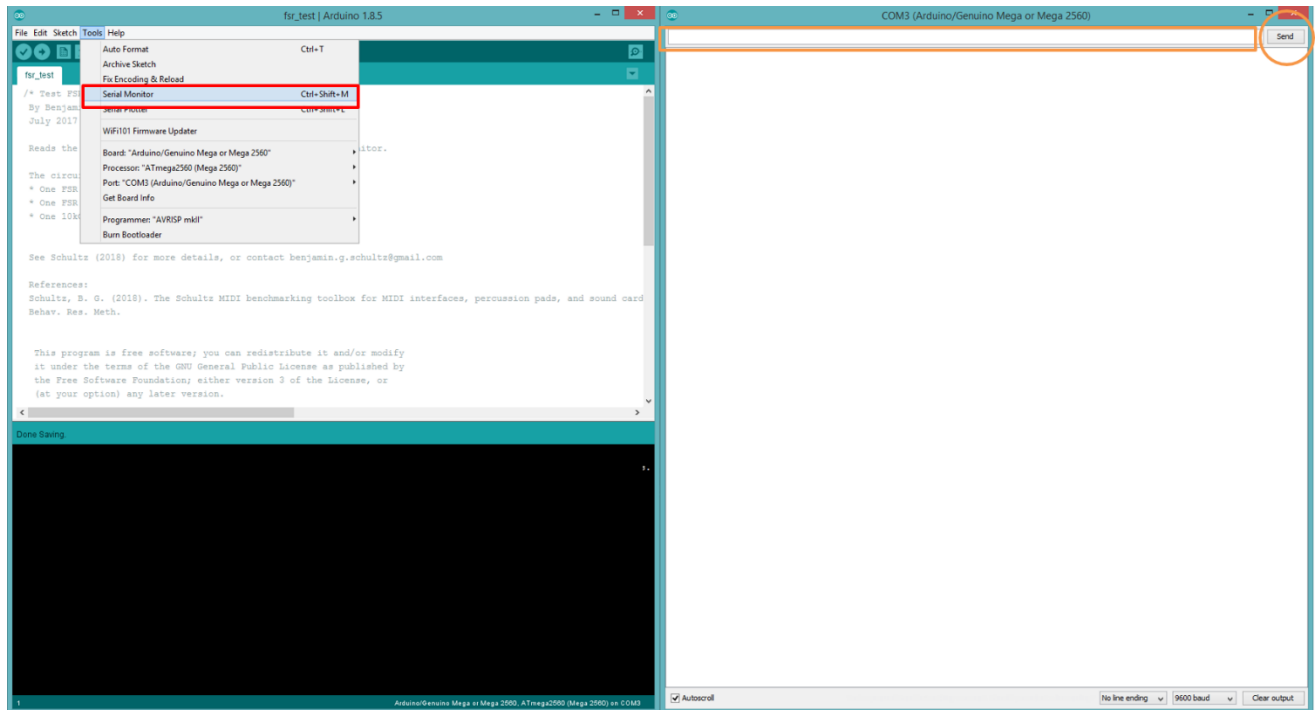
Optionally, you can instruct Mac or Linux to always map the Arduino to a particular port, by writing an “udev” rule. Please refer to an “udev” tutorial for this option, since it is beyond the scope of this manual.

### 2.2.2 Uploading (flashing) code to the Arduino

To benchmark MIDI devices with the Arduino, you will need to upload the SCiBuT codes 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 clues as to what the error is (see section [4.2](#)). Please ensure that the code is uploaded to the correct Arduino (and on the correct Serial communication port or COM port).

### 2.2.3 Arduino Serial Monitor

To open the serial monitor, go to Tools → Serial Monitor, or press Ctrl + Shift +M simultaneously (see Figure 1).



*Figure 1.* Arduino interface (left panel) and serial monitor (right panel). The red rectangle show the command to open the serial monitor. The orange rectangle shows where commands can be sent to the Arduino to start “Send” tests. The orange circle shows the button you can press to send the command to start the test (or you can press enter after typing).

### 2.3 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 Python script. 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.

In Ubuntu, this software installation step can be done using the following command: `sudo apt-get install python python-serial`. This step also installs the Python serial module, so you Ubuntu users can skip sections 2.3.1 and 2.3.2.

### **2.3.1 Download the pySerial module**

To read data from the Arduino, 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).

### **2.3.2 Installing the pySerial module**

#### **2.3.2.1 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
```

#### **2.3.2.2 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
```

### **2.3.2.3 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/>

### **2.3.3 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 SCiBuT, only pyserial is required.

### 3. Hardware Configuration

Here, I give you step-by-step instructions for how to build the visual sensor and connect it to an analog input box and an Arduino. I then show you how to test the equipment to make sure everything works as we go along. This should help with troubleshooting later. As a general rule, the colouring of my wires will consistently follow this convention: red is connected to power pins (5V), black is connected to ground pins (GND), and green and purple are connected to analog inputs for the visual sensors (A0 to A5).

#### 3.1 Photodiodes

This section describes how to build the visual sensor and connect it to an analog input box or an Arduino microcontroller.

##### 3.1.1 What to buy

For the basic setup, you will need to purchase two photodiodes, two breadboards, some hook up wire, and two  $470\Omega$  resistors with the colours gold, brown, purple, and yellow (see Appendix A). If you do not have much (or any) experience with soldering, I recommend you either use the described method which avoids soldering altogether but produces a device that requires a great deal of care when handling (to ensure wires to not come loose).

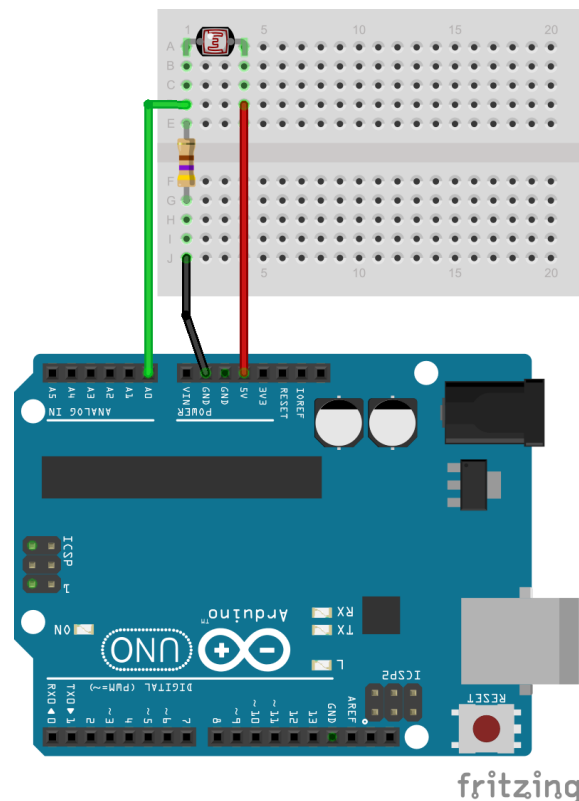
##### 3.1.2 Setup and schematics

While the Analog input box and/or Arduino is OFF (the power and USB are not connected), arrange the cables as follows (see Figure 2):

1. Connect the photodiode to the breadboard making sure each pin is in a different row (each “row” contains 5 pins, see Figure 2). You may need to cut the wires of the photodiode if you wish to achieve a low profile.
2. Connect a red wire between the breadboard row containing the left pin of the photodiode and the 5V pin.
3. Connect a green wire from the same breadboard row as the right photodiode pin to the first analog input pin (A0 of the Arduino). This wire will be used to record the brightness of the left top corner of the computer screen.
4. The  $470\Omega$  resistor acts as a voltage divider between the ground (GND) pin and the incoming signal from the photodiode. Connect one end of the resistor to the same breadboard row as the green wire, and the other end to a different breadboard row.
5. Connect one end of a second 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.

6. If you are using two visual sensors, repeat steps 1 to 5 but use a purple wire instead of green and connect the purple wire to a second analog pin (A5 of the Arduino).

Congratulations! You have just connected the photodiode(s) to your analog input box or the Arduino. Now we should test that everything works.



*Figure 2.* Solderless design for the photodiode with the Arduino. Wires should be long enough to reach the monitor. Figure produced using fritzing (Knörig, Wettach, & Cohen, 2009).

### 3.1.3 Testing the Photodiode

Once you have connected the photodiode(s) to the Arduino or analog input box, we can test that the visual sensor is working.

#### 3.1.3.1 Testing the Photodiode with an analog input box

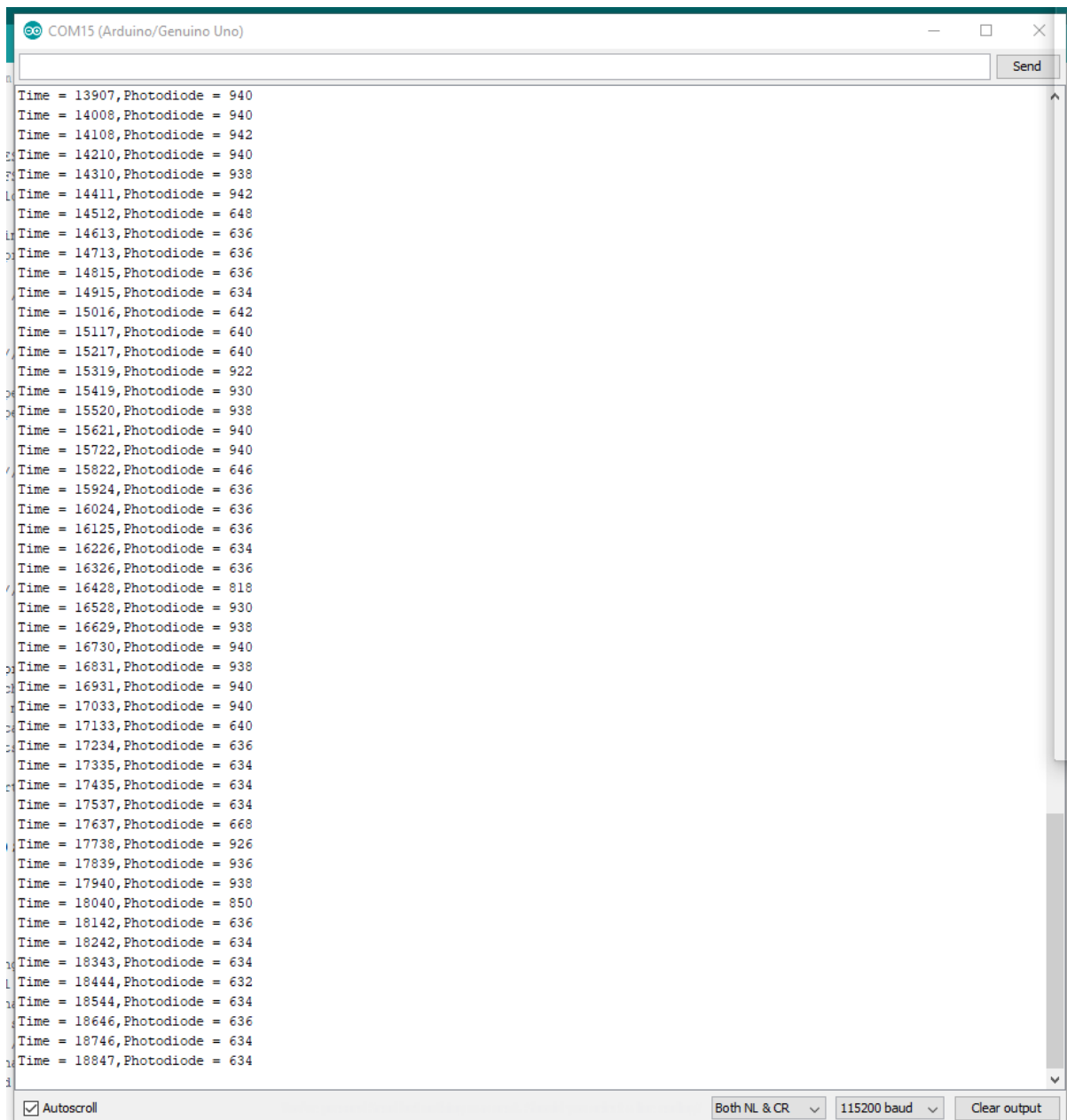
As there are multiple types of analog input boxes and associated software packages, you will need to refer to the specific user manual in order to perform each step. Here I provide general instructions that **should** apply to most devices. Ensure that the

analog input box is connected, switched on, and is syncing with any interface used to connect the analog input box to the computer. If using EEG, please ensure your amplifier is powered and switched on. In your software package, ensure that the physical channel connected to the visual sensor is being read in the workspace. These channels are often referred to as “Auxiliary channels” and are listed after any other channels (e.g., EEG, ECG, EMG). It sometimes helps to include all physical channels when searching for the channel representing your electrode. The simplest way to test the photodiode is to cover and uncover it with an item (e.g., a box or your hand), preferably without moving the cables. You can also shine a torch into it or play one of the flicker videos you can construct using the MATLAB script called *SCiBuT\_makeFlickerVid.m* (see section 4.1). As the amount of light entering the photodiode increases and decreases, the voltage presented on screen in your software package should also increase and decrease for that channel.

### 3.1.3.2 Testing the Photodiode with the Arduino

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 “*scibut\_test\_sensor.ino*” sketch using the Arduino software. Upload the sketch as shown in section [2.2.2](#) and open the “Serial Monitor” as shown in section [2.2.3](#). You should see something like the output in Figure 3. Cover and uncover the photodiode with an item (e.g., a box or your hand), preferably without moving the cables. You can also shine a torch into it or play one of the flicker videos you can construct using the MATLAB script called *SCiBuT\_makeFlickerVid.m* (see section 4.1). This is how continuous data is read into the Arduino from the photodiode. The values will only be reported when there is change in the signal (to prevent buffer overflow on the USB).



*Figure 3.* The Arduino serial monitor showing a series of frame changes recorded by the Arduino. In this case, our values range from 632 (representing black) to 940 (representing white). Your values may vary depending on supplied voltage and selected resistor.

## 4. MATLAB Code

The MATLAB scripts already contain documentation within each script and an example. I briefly describe the scripts here.

### 4.1 MATLAB Code to make flicker videos

We have added a script to allow users to create their own flicker videos to examine framerate changes on their own system. The user can select the video duration, frames per second, pixel dimensions, size of the cigarette burn markers, and the number of cigarette burns. The script is called *SCiBuT\_makeFlickerVid.m* and contains documentation within the file. We have also added a script called

### 4.2 MATLAB Code to add visual markers

The SCiBuT contains a script to add cigarette burn markers to your own videos. Running the script *SCiBuT\_addCB.m* with a video filename (including directory) will add cigarette burn markers (up to 4) to your video. We have also included a script that selects all video files within a folder and adds cigarette burn markers, called *SCiBuT\_addCBall.m*.

### 4.3 MATLAB Code to extract onsets/offsets

There were three dependent variables examined in Schultz et al. (submitted): The average framerate, the coefficient of variation of the framerate, and the number of missed frames. The example script *SCiBuT\_example.m* extracts all of this information for you. Note that to extract the number of missed frames, you require at least two visual sensors. If you only have one visual sensor, you could make the assumption that any frame that occurs more than one and a half periods from the expected onset time was a missed frame but it might also reflected a late frame.

This test provides summary statistics for the framerate by examining the veridical onset and offset times measured by the sensor. This test can be performed using only one visual sensor but it would be difficult to determine if a frame was late or skipped/dropped. The example script uses a test dataset recorded by an Arduino to extract summary statistics of the framerate.

The MATLAB code first imports the time series data from the Arduino. Any time series data with a consistent sampling rate can also be examined, for example, data from an analog input box or oscilloscope. In some cases, additional filtering (e.g., high-pass filters) might be required but this is outside the scope of the present

manual. Please visually inspect your data first to ensure you have removed drift and artefacts (e.g., extreme voltages) from your data.

The MATLAB code then range normalizes your data so a black marker represents values closer to 0 and a white marker represents values closer to 1. Then, the onsets and offsets are extracted. To avoid issues related to rise and fall times, the code looks for the first moment of change after crossing the threshold (user defined; somewhere around the midpoint of 0.5 is best). We extract the periods by taking the differential between consecutive onset and offset times. For each period, we calculate the frames-per-second (FPS) by dividing 1 by the period. We then calculate the mean, standard deviation, minimum, and maximum FPS as well as the coefficient of variation defined as the standard deviation divided by the mean.

Finally, the MATLAB code determines missed frames by transforming onsets and offsets into binary and returning the cumulative count to ensure values are sequential. The code returns the number of times the values did not follow the linear sequence of expected values, suggesting a missed frame.



## 5. Troubleshooting

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

### 5.1 Problems with the Arduino GUI

I haven't had any problems with the 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 and rows, and that no wires are connected that shouldn't be. See throughout Section [3](#) 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 when uploading your code or sending a command. Check again by following the instructions in section [2.1.1](#).
5. 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. I 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. If you have gone through all of these steps and you are still having issues, please email me with the specifics of the problem, with error messages and system information (e.g., operating system):  
[benjamin.glenn.schultz@gmail.com](mailto:benjamin.glenn.schultz@gmail.com)

## 5.2 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 scripts. The serial monitor can be accessed in the Arduino IDE by going to the “Tools” tab and clicking on “Serial Monitor”. I have included parts of the script that can be read in the Arduino serial monitor, just in case you are having difficulties. These sections are flagged by “DEBUGGING” and “END DEBUGGING” in the Arduino code. If you remove the “//” at the beginning of the lines between “DEBUGGING” and “END DEBUGGING” then you can use the serial monitor to read the output from the Arduino. Note that the “SCiBuT\_Read” scripts contain two debugging sections to let you know if the correct command is received, and then the note and velocity bytes. The serial monitor is also useful for receiving errors that might arise while using the SCiBuT. 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 (this should remain at 9600, as defined in the “SETUP” section of the code).

## 5.3 The Python Scripts Aren’t Working!

If everything is working with the Arduino and you can see data in the serial monitor, you may still get an error in the Python script. The two usual suspects are:

1. PySerial has not been installed (see sections [2.3.1](#) and [2.3.2](#))
2. The COM port needs to be defined on line 50 of the Python script. In this case, check the COM port of the Arduino using the GUI (or device manager) and change line 50 to:

```
comm_port = "COM#"
```

Where # is replaced with the number provided.

If you experience another error, please contact me with the error code:

[benjamin.glenn.schultz@gmail.com](mailto:benjamin.glenn.schultz@gmail.com)

## 5.4 The MATLAB Scripts Aren’t Working!

That’s not good. Please double-check that you followed the instructions and, perhaps, contact someone who is a MATLAB expert (if you aren’t one already). If that fails (or is not an option), please email me with the error code and MATLAB version: [benjamin.glenn.schultz@gmail.com](mailto:benjamin.glenn.schultz@gmail.com).

## 5.5 I’m having a problem you haven’t thought of!

Inconceivable! If you do, however, come across a problem or issue that I have not documented here, please let me know. I would gladly try my best to get you up and running. Please email me: [benjamin.glenn.schultz@gmail.com](mailto:benjamin.glenn.schultz@gmail.com) with your name, OS, Arduino version, MATLAB version, and a description of your problem.

## **6. Citing this software**

If you use any of the scripts or codes that have been shown in this manual or from the website ([www.band-lab/smilib\\_toolbox/](http://www.band-lab/smilib_toolbox/)), please cite the associated article:

Schultz, B. G., Biau, E., & Kotz, S. A. (submitted). An open-source toolbox for measuring dynamic video framerates and synchronizing video stimuli with neural and behavioral responses. *Nature Methods*.

### **6.1 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, I would be happy to send you a draft of the paper until it is accepted.

### **6.2 Contributions**

This project has followed on from a previous collaboration between Benjamin Schultz and Floris van Vugt (Schultz & van Vugt, 2016). I (Benjamin Schultz) have developed these scripts through the very useful forums available on the Arduino website ([www.Arduino.cc](http://www.Arduino.cc)), several Instructables pages ([www.instructables.com](http://www.instructables.com)), and some trial and error. I am indebted to Floris and these forums for providing valuable insights into how these systems work and the best way to communicate between different human interface devices.

## 7. References

- Adafruit (2015). Adafruit Wave Shield for Arduino kit,  
<http://www.adafruit.com/product/94>
- Arduino (2015). <http://www.arduino.cc/>
- 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. (2016). Tap Arduino: An Arduino microcontroller for low-latency auditory feedback in sensorimotor synchronization experiments. *Behavior Research Methods*, 48(4), 1591-1607.
- 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 MEGA or Due	<a href="http://store.arduino.cc/">http://store.arduino.cc/</a>
Square force sensitive resistor (FSR 406)	<a href="http://www.interlinkelectronics.com/standard-products.php">http://www.interlinkelectronics.com/standard-products.php</a>
TRRS 3.5mm Headphone Jack Breakout	<a href="https://www.sparkfun.com/products/11570">https://www.sparkfun.com/products/11570</a>
Resistors  10k $\Omega$ , brown, black, orange, gold  220 $\Omega$ , red, red, brown, gold  470 $\Omega$ , yellow, violet, brown, gold	<a href="https://learn.sparkfun.com/tutorials/resistors/types-of-resistors">https://learn.sparkfun.com/tutorials/resistors/types-of-resistors</a>
Female MIDI Connector	<a href="https://www.sparkfun.com/products/9536">https://www.sparkfun.com/products/9536</a>
Mini Breadboard	<a href="https://learn.sparkfun.com/tutorials/how-to-use-a-breadboard">https://learn.sparkfun.com/tutorials/how-to-use-a-breadboard</a>
USB 2.0 Type A to Type B (NOT the mini USB connector)	<a href="https://www.sparkfun.com/products/512">https://www.sparkfun.com/products/512</a>
Optocoupler	<a href="https://www.digikey.com/product-detail/en/vishay-semiconductor-opto-division/6N138/751-1263-5-ND/1731496">https://www.digikey.com/product-detail/en/vishay-semiconductor-opto-division/6N138/751-1263-5-ND/1731496</a>
Diode	<a href="https://www.sparkfun.com/products/8588">https://www.sparkfun.com/products/8588</a>
Hook up wire (kit)  Wire colour unimportant	<a href="https://www.sparkfun.com/products/124">https://www.sparkfun.com/products/124</a>
Screw terminal  (3.5mm, 2-pin)	<a href="https://www.sparkfun.com/products/8084">https://www.sparkfun.com/products/8084</a>
Break away headers  (Male)	<a href="https://www.sparkfun.com/products/116">https://www.sparkfun.com/products/116</a>

## Appendix B: Electronics Vendors

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

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