A low-cost, highly accurate environment for imaging neurons in different experimental settings

Abstract

Introduction

Neuronal imaging is a burgeoning technique that demands temporal fidelity. A new challenge is the development of techniques that allow for concomitant execution of different tasks and experimental paradigms in a way that is synchronized with imaging. Of recent interest include examining the motor output of mice while imaging from relevant parts of the striatum (Barbera et al., 2016 and Klaus et al. 2017), and imaging the hippocampus during classical conditioning (Muhammad et al, 2015). We demonstrate that these two experimental configurations can easily be achieved using a Teensy 3.2 microcontroller.

*General requirements*

Experimental designs in imaging require several components in order to be successful. First and most importantly, behavioral data must be precisely organized with respect to imaging data. Capturing images has strict temporal requirements due to the fact that fluorophores must first be exposed to an LED for a fixed amount of time before a camera can capture the emitted fluorescence. A common design is to set up a CMOS or other imaging device to capture a frame every time it sees the rising phase of a digital pulse from the device responsible for organizing and synchronizing the experiment. Therefore, substantial jitter in digital pulse delivery can cause potentially substantial frame loss. That is, if the camera has not finished with the previous imaging cycle, it could skip a frame if the next pulse signal occurs too early. With respect to imaging paradigms based on classical conditioning, one also needs to worry about a mouse trained to respond to a conditioned stimulus: repetition of stimulus and response must occur in a highly regular temporal fashion.

Secondly, the experimental setup must be easy to manipulate or alter. Technical skillsets vary widely in the field of neuroscience, and to be adapted widely, experimental designs must accommodate these widely varying backgrounds. It is infeasible and inefficient to rely on a technician every time one must subtly tweak or disturb an experimental paradigm. Ideally, the componentry of an experimental setup would enable a user to quickly translate or implement an idea they have in mind, and be simple enough to encourage the user to build novel experimental designs instead of conforming to preexisting designs. Experimental setups should accelerate and not impede the pace of research and discovery. The Arduino programming environment, which Teensy utilizes, is simple to learn for anyone with any programming backgrounds, as explained in depth by (D'Ausilio, 2012). Digital output and timing libraries are also easy to learn and implement.

Finally, the experimental setup should be both widely accessible and open-source. Current environments and programming environments can be exceedingly expensive. As we see in Table 1, the Teensy 3.2 itself costs only $19.80 from the manufacturer. The most expensive experimental component that we use in either setup is the ADNS-9800 sensor, which costs only $27.50. The Arduino and added-on Teensyduino programming environments are free.

Cost can be prohibitive; therefore, if an open-source programming or design environment existed but were expensive, this would preclude wide-spread contribution of new software libraries or hardware components to the existing system by pricing out hobbyists. For example, in our implementation of a motion-sensing calcium imaging paradigm, we utilize the ADNS-9800 sensor, which is produced by a small company (Jack Enterprises, LLC) in Cookeville, Tennessee. This sensor affords us easy and affordable access to a high-speed, high-fidelity gaming sensor. Open-source products potentially offer faster, highly parallel development by taking advantage of the global village. Cost improves accessibility, and accessibility is necessary to maximize the effect of an open-source environment. Even if money is not an object to academic audiences, the lower the cost of an item, the more readily hobbyists will adopt the product. As more and more hobbyists do so, we will see the development of new open source libraries accelerate, which does in fact influence the academic community.

*Motor acquisition*

There are a number of ways in which people have attempted to observe motor output while imaging. In one particular technique, experimenters mount a fluorescence microscope on the head of a mouse, and allow the mouse to move freely while recording activity via video (Barbera et al. 2016) or via video in addition to an accelerometer (Klaus, et al., 2017). However, resting a microscope on the head of a mouse restricts its normal range of movement for the mouse, limiting its peak velocity and introducing a confounding variable to the experiment. For example, bearing additional weight recruits more muscle fibers and potentially supportive architecture, which could blur distinctions between neural representations of high and low motor patterns, particularly in motion-related regions of the brain such as the striatum.

Another technique utilizes a “three-dimensional treadmill” setup, initially proposed by (Dombeck, Khabbaz, Collman, Adelman, & Tank, 2007) and utilized widely elsewhere (Aranov & Tank, 2014). In this setting, the mouse is fitted with a head plate and imaging window, and is suspended atop a Styrofoam ball that is supported by compressed air (Figure 1). This type of imaging offers small image jitter primarily in-plane, which is advantageous because it can easily be corrected by standard cross-correlation-derived motion-correction methods. It also offers a setting in which mouse must apply similar forces to begin or to terminate a motor sequence as it would in a freely-moving setting (Dombeck, Khabbaz, Collman, Adelman, & Tank, 2007). Therefore, the mouse is able to move at normal velocities. Generally, two computer mice are fit at the equator of the styrofoam ball at an angle of 90 degrees, which provides the experimenter with linear movement in the X-Y plane, as well as rotational information. Most of these techniques utilize LabView to obtain voltage readings from the computer mice (Aranov & Tank, 2014) (Dombeck, Khabbaz, Collman, Adelman, & Tank, 2007) which, though a comprehensive piece of software, is expensive and proprietary. Based on our own testing, implementing high-level MATLAB implementations of TTL pulse-based data acquisition using a National Instruments data acquisition board led to temporal delays and jitter. As described above, imaging needs a platform that is low-cost, scalable, and had high temporal fidelity. Our system achieves this.

In an effort to improve the ease with which users can adopt our motion-sensing paradigm, we have built and included classes and drivers that abstract away the complexity of interacting with the ADNS-9800 sensors. We think that this is the optimal way to observe motor data collected from a mouse. The ADNS-9800 sensors are highly sensitive and have high maximum sampling rates, with a maximum read rate of 12000 frames per second, and 8200 counts per inch resolution (<https://datasheet.octopart.com/ADNS-9800-Avago-datasheet-10666463.pdf>). Further, accumulated displacements can be stored in the sensors between readings, because ADNS-9800 sensors store motion data in 16 bits instead of the standard 8 bits. Therefore, one does not need to worry about polling sensors as frequently as one would need to with a standard computer mouse.

*Teensy 3.2*

The Teensy 3.2 (<https://www.pjrc.com/store/teensy32.html>) is a less well-known microcontroller with several advantages compared with the Arduino, which has also been utilized to design experiments in the realm of neuroscience (D'Ausilio, 2012), (Chen & Li, 2017), (Micallef, Takahashi, Larkum, & Palmer, 2017). First, its processor, an ARM Cortex-M4 MK20DX256, has a much higher clock rate than the Arduino (72 MHz vs 16 MHz), allowing for faster and theoretically more precise data acquisition. Second, it has an output voltage of 3.3 Volts, compared to the Arduino’s 5 Volt output. This offers a small practical advantage for our motor experiment, as activating 5 Volt mode on the ADNS-9800 sensors requires additional soldering and modifications to the sensors. Third, this device is capable of utilizing “IntervalTimer” objects for microsecond-level precision in calling different functions using interrupts. This allows us to reliably acquire velocity estimates from our sensors at 20 Hz or at any other reasonable imaging speed. It also has the very useful “ellapsedMicros” and “ellapsedMillis” libraries built in to the Teensyduino library. Though these can be downloaded separately for the Arduino, but come preinstalled in the Teensyduino library. They enable the user to determine the time, either to microsecond or millisecond accuracy, that has passed since a corresponding variable is initialized. It also offers true analog output, while the Arduino only offers pulse-width modulation for simple audio. In fact, a comprehensive audio library that runs at 44.1 kHz is built for the Teensy (<https://www.pjrc.com/teensy/td_libs_Audio.html>). Thus, this device is capable of generating sounds, even complex sounds, while simultaneously executing other tasks. Finally, the Teensy programming environment utilizes the Arduino’s programming environment. Therefore, it can utilize all of the crowd-sourced functionality of the Arduino’s massive user base, while also taking advantage of the unique features that the Teensy itself offers.

Here, we introduce two specific implementations of experimental designs geared toward neuronal imaging, implemented via a Teensy 3.2 microcontroller in conjunction with several simple code scripts and a library for the ADNS-9800 sensors. We thereby demonstrate the ease and usefulness of adopting Teensy 3.2-based designs for future imaging experiments, and for fulfilling the requirements of temporal accuracy, ease of use, flexibility, low cost and high accessibility.

**Methods**

*Motor acquisition experiment*

The overall design for this experiment is shown in Figure 1A. Two ADNS-9800 gaming sensors are attached at the equator of a 3D-printed half-sphere in which a large, buoyant Styrofoam ball is floating. These sensors lie at an angle of approximately 75 degrees from one another. This setup mimics that of (Dombeck, Khabbaz, Collman, Adelman, & Tank, 2007). The precise wiring to both sensors is demonstrated in Figure 2A. To compute linear velocity, we use the y-readings of both sensors. We can compute rotation using the x-readings if we wish. These two sensors are attached via simple serial peripheral interface (SPI) connections to the Teensy, the details of which can be seen in Figure 2. One can implement this design with inexpensive jumper wires or another type of wire and minimal soldering. Finally, the Teensy is itself connected to a full computer using a USB-microUSB cable.

Due to the complexity of extracting software from these sensors, we wrote simple classes and functions that are freely available on Github and abstract the complexity to a user-friendly level. In the current setup, we implemented a version of data-extraction where every 50 milliseconds, the “motion burst” register of each sensor is read, and the accumulated displacement over the previous 50 milliseconds in the x and y directions is acquired. Simultaneously, a digital “on” pulse is sent out of a digital pin using the DigitalIO library (<https://github.com/greiman/DigitalIO>). This allows us to use the functions “fastPinMode” and “fastDigitalWrite’, for example, which reduce the latency introduced by turning pins on, off, or setting their “mode” (input or output). A main function that directs both of these tasks was called in precisely timed intervals using the “IntervalTimer” function. Instead of using the default Arduino programming environment to upload our code to the Teensy, we used PlatformIO (<https://platformio.org/>), an add-on to the widely-used Atom text editor (<https://atom.io/>). This allowed us to easily build and upload our multi-folder library to the Teensy.

In order to begin experiments with the Teensy, after the main script was uploaded, we wrote a simple MATLAB-based graphical user interface that can be used on a desktop or laptop connected via a USB to the Teensy. In principle this graphical user interface could be written in Python or any other programming language. Using this, the user can enter the length of the experiment and the frequency of data acquisition. This frequency will determine the frequency with which digital pulses are sent to notify an external device such as a CMOS camera to capture am image, for example, and also determine the frequency with which accumulated motor information will be recorded by the PC. The PC or laptop sends this information over a serial connection to the Teensy utilizing the bidirectional microUSB-USB cable.

In a proof-of-concept experiment (Figure 3), we recorded an approximately 10 minute long session of a mouse running on a 3 dimensional treadmill (styrofoam ball floating on air), and data was acquired 20 Hz concomitant with digital pulses that could be used to trigger a camera or a different device. Sensors were placed in a 3D-printed half-sphere at an angle of approximately 75 degrees from one another at equal heights, slightly below the equator of the ball. The mouse’s speed was computed using the y-coordinates of each mouse sensor, and the total distance travelled at any one time point was computed in the following way:

Where yR and yL are the y readings from the left and right sensors, and is the angle between the two sensors (75 degrees). Velocity was computed as the distance divided by the time between two adjacent frames. Times and distances travelled were recorded by the Teensy 3.2, and the timings of digital pulses were measured by an external device at 3051.76 Hz (Tucker-Davis Technologies RZ5D). Pulses were considered to be logical ones at the first time point measured where the input voltage exceeded 1 Volt.

*Classical conditioning experiment*

To illustrate another simple experimental design wherein the Teensy 3.2 can be used to control four devices simultaneously and output a sound, we created a tone/light-puff setup and wrote the corresponding script. The general setup is shown in Figure 1B. In this design, a head-fixed mouse would be exposed to a 9500 Hz tone concomitantly with a light stimulus. After, the mouse receives a puff of air in its eyes. The goal was to train the mouse to blink upon exposure to the unconditioned stimuli.

In order to amplify the sound to a suitable volume, we added a “prop shield” to this design. The prop shield is a very affordable, easy-to-use add-on that is capable of amplifying the analog output signal (shown in Figure 2B as pin A14). A less expensive option is also available for $8.40. In addition, if stereo output were desired, the manufacturer also offers a true audio shield (<https://www.pjrc.com/store/teensy3_audio.html>) that is capable of stereo output.

To attach the prop shield to the Teensy 3.2, we used 14x1 double insulator pins, and then fed the output to a speaker, as demonstrated in Figure 2B. The prop shield can power speakers with resistances up to 8 ohms (https://www.pjrc.com/store/prop\_shield.html). We also directed digital outputs from the Teensy to activate a light concomitant with the sound, and a puff as an aversive stimulus following each sound/light combination. Meanwhile, digital pulses were programmed to occur during every frame, which could be used to trigger image captures, for example. The “IntervalTimer” library was used in order to reliably orchestrate TTL pulses and the multiple output devices. This library utilizes interrupts in order to precisely call a specific function at prespecified time intervals. The speaker, camera, puff, and light source can be attached to the microcontroller using simple coaxial cables with SMA connectors, as shown in Figure 1A. The same programming environment (PlatformIO on top of Atom) was utilized, and functions such as “fastPinMode” and “fastDigitalWrite” were utilized to decrease latency. A main function was called every loop using the aforementioned “IntervalTimer” function with precise timing.

Again, in order to begin experiments with the Teensy, we wrote in MATLAB a simple graphical user interfaces that can be used on a desktop or laptop. With this, a user can enter the length of the each trial and the number of trials. The PC or laptop sends this information over a USB connection to the Teensy, which in turn reports information about the experiment, in particular the frames during which the tone is on, the puff is on, or the light is on. In our proof-of-concept experiment (Figure 3), the puff, light, and camera trigger pulses were all attached to and were recorded by the same external device (TDT RZ5D) at 3051.76 Hz. We performed a mock-recording consisting of 10 trials of 15 seconds length each, where sound and light output pins were turned on 11.1 seconds into each trial for 350 ms, and the pin used to generate the aversive puff stimulus was turned at 11.7 seconds into each trial for 100 ms. Output from the puff, light, and camera pins were recorded by an external device at 3051.76 Hz. For data analysis, the data point when the voltage first exceeded 1 Volt was considered to be the time at which the particular output was first active. The data point when the voltage first dipped below 1 Volt was considered to be the end of the stimulus.

**Results/Discussion**

We constructed two separate and commonly utilized experimental setups both built upon a Teensy 3.2. In the first (Figure 1Ai and 1Aii), we constructed a device that monitors and records motor data at a fixed interval, which is capable of simultaneously delivering highly regular, brief digital pulses to an external device such as a CMOS camera. As shown in Table 1, the cost of specialty components for this experimental design is quite low. Other commonly used components such as wiring, solder and wire strippers and crimpers are also needed, but are widely available and should in many cases such as a lab setting already be available.

*Motion tracking using the ADNS-9800*

Here we introduce a system for imaging and simultaneous motion three-dimensional treadmill tracking that necessitates only a Teensy 3.2 microcontroller and two ADNS-9800 laser motion sensors. In order to provide the end user with as simple a setup as possible, we designed drivers and a library that users can use to obtain various streams of data from these sensors. In particular, we read displacements picked up by the sensors and convert them directly to micrometer displacements using the internal calibration of the sensors. Conversions to imperial or metric distances can also be implemented via our ADNS-9800 library. Using different registers of the ADNS-9800 sensors, users can also query the sensors for a host of other features, for example, the surface quality that the sensor detects. This could be useful if the ball or surface that the sensor is measuring changes its position slightly with respect to the surface. Each register in the datasheet for the ADNS-9800 is included in the ADNS9800 library, explanations for which are available online (https://datasheet.octopart.com/ADNS-9800-Avago-datasheet-10666463.pdf).

Therefore, little must be done besides implementing the proper wiring in order to get such a design up and running, particularly if one is interested mostly in recording accurate x, y, and rotational displacements, which we have already implemented directly in the code. Proper wiring is demonstrated in Figure 2A. The connections demonstrated using dotted lines can be replaced with jumper wires or sturdier, longer lasting wire. Anecdotally, jumper wires appear to become unreliable after a short amount of time, and so sturdier wire is preferred. This system offers an affordable, modular, open-source method of tracking mouse movement with high fidelity, temporal accuracy and without introducing confounding experimental variables. As can be seen in Figure 3A, the velocity that we calculate falls into the range of previously reported mouse velocity with similar setups (see, for example, (Dombeck, Khabbaz, Collman, Adelman, & Tank, 2007)), and we are capable of seeing quite a bit of variation in the mouse’s motor output. In Figure 3B, we also see that digital pulses administered at 50 ms increments closely track the theoretical times, biased in slope by an exceedingly small amount (approximately 29 microseconds per sample).

*Classical conditioning*

In the second experiment (Figure 1B and 2B), we constructed a device capable of running a simple classical conditioning experiment, where we can train a mouse to blink in response to a tone and light exposure using a puff of air as an unconditioned aversive stimulus.Our design of a classical conditioning experiment mimics the setup previously reported (Mohammed, et al., 2016). As previously described, a mouse is gradually trained to blink after seeing a light and hearing a sound, via a “puff” that is consistently delivered following exposure to both light and a 9500 Hz tone. Imaging can be performed simultaneously by turning on and off a given pin during each frame using the “IntervalTimer” function, the rising phase of which a camera or other device can use as an indicator telling it to capture an image. In a mimic experiment, we recorded the timings of each of these triggers and compared them to the theoretical timings with samples spaced at exactly 50ms apart, as shown in Figure 4. First, as with the motion experiment, we looked at the individual “camera-trigger” timings. Like the motion experimental design, the measured timings were very similar to the theoretical timings, biased downward by approximately 3e-05 seconds per sample. We looked at light onset and offset, and puff onset and offset in Figure 4B as well. All were consistent with the theoretical timings of these events (slope close to 1 in all cases), showing a consistent upward bias of only 3e-05 seconds per sample as well. We not that, adjusted for the length of time, our timing bias is comparable to that reported by (D'Ausilio, 2012) in various Arduino experimental designs at approximately 0.6 milliseconds per second (3e-05 per sample / 0.05 seconds per sample = 6e-04 per second).

**Conclusion**

Here, we demonstrate two inexpensive and highly accurate experimental paradigms both constructed around a Teensy 3.2 microcontroller. In the first, we have designed and implemented a library capable of recording motor output from ADNS-9800 gaming sensors without the need for outside calibration. The Teensy is capable of performing this task while sending temporally accurate digital pulses out of another digital pin. This would be particularly useful in an imaging paradigm, where a camera needs to be triggered with highly precise timing.

We also demonstrate a setup built to implement a classical conditioning paradigm. This illustrates the ability of the Teensy to orchestrate many different classes of output simultaneously with high temporal accuracy, and also highlights the ability of this device to simultaneously produce an analog output, in particular to generate a sound, while performing other actions. One other major advantage of the Teensy 3.2 over other microcontrollers such as the Arduino is the fact that it can output a true analog signal, whereas the Arduino Uno, for example, is capable only of outputting pulse-width modulated signals. This opens a venue for many experimental additions, particularly the addition of sound, without the need of extra devices such as resistors and capacitors to create an analog signal. Rather, the Teensy 3.2 simply needs to be soldered on to a prop shield, and no in-depth knowledge about electronics is necessary.

The only potential limitation of our system that we saw was the slight timing drift of the Teensy, on the order of 3e-05 seconds per sample (or approximately 30 microseconds per 50 millisecond sample). This drift is linear in nature, however, so it would be easy to calibrate out. We note as well that standard error of our measurements across all linear models were very small: on the order of tens to hundreds of nanoseconds. This error is likely overestimated as well, as the sampling rate of our measuring device is only capable of reading at a frequency of 3051.76 Hz. The precision and utility of the Teensy microcontroller, in conjunction with the custom motion sensor library that we have developed for the ADNS-9800 sensors, make this a user-friendly, easily adaptable tool for various experimental designs in the neurosciences.

**Figures**

**Figure 1.** Diagrams of the two experimental device setups, a floating, 3D treadmill with two sensors for recording motor output (A) and a tone/light and puff classical conditioning setup. **A** This experimental design consists of a Teensy 3.2 connected to two ADNS-9800 sensors and a CMOS camera, via serial-peripheral interfaces and a coaxial cable via SMA connectors, respectively. Every 50 milliseconds, a digital pulse triggers the CMOS camera to capture an image while simultaneously acquiring motor data from both ADNS sensors and sending them via a USB to a PC. The PC initiates each experiment by sending serial data consisting of the length of the experiment and imaging frequency to the Teensy. **B** This experimental design constitutes a classic classical-conditioning paradigm. The user specifies via MATLAB or via a different interface the length and number of experimental trials. This information is sent via a USB to the Teensy 3.2, which initiates the experiment. In each trial, the Teensy initiates a 9500 Hz tone at 44.1 kHz while turning on a light. These stimuli are followed by an air puff, also delivered via the Teensy. In order to generate a sound loud enough for the speaker, the Teensy is soldered to a prop-shield, which contains an amplifier. The Teensy 3.2 sends time stamps, trial, and stimulus information via the USB back to the PC.

**Figure 2. A.** A schematic demonstrating the wiring connections between a Teensy 3.2, prop shield, and an external speaker. Dotted lines indicate solid connections. All connections between the Teensy 3.2 and prop shield were made using 14x1 double insulated pins, and the output to the speaker from the prop shield was made using regular wire and a coaxial cable. Some extraneous and unused pins on the Teensy and the prop shield were not included in this diagram. **B.** A schematic demonstrating the wiring of Teensy to two ADNS-9800 sensors via serial peripheral interface connections (SPIs). Solid dots at intersections between lines indicate connections. Some unused pins on the Teensy 3.2 were not included in this schematic.

**Figure 3.** **A.** Part of a sample 10 minute recording session during which a head-fixed animal was allowed to run on the three-dimensional treadmill. Shown in **Figure 1A**. Its average speed is 7.1 + 6.9 cm/s, with a maximum velocity of 47.0 cm/s, within ranges reported elsewhere. **B.** Times of digital pulses sent by the Teensy 3.2 as measured internally by the Teensy, vs times of the digital pulses as measured by an external device. Data are jittered for better visualization. The linear model estimates a slope of 1.000028927 + 5e-09 (t(11997)= 2.0381e+08, p < 0.001, R2=1; intercept = 1.07e-04 + 0.02e-04, t(11997) = 63.243, p < 0.001), indicating an excellent fit and very nearly a 1:1 correspondence of time stamps.

**Figure 4.** **A.** Timing of the digital pulses as measured by the Teensy 3.2 in the tone/light-puff setup versus timing as measured by an external device. These measurements have a correspondence near 1:1 (R2=1, slope: 1.00003326 + 4e-08, t(3006)=2.52e+07, p<0.001). **B.** Timing as measured by the Teensy 3.2 vs timing measured by an external device over the course of ten trials for the beginning and end points of conditioned stimulus (light) and the beginning and end points of the unconditioned stimulus (puff). All exhibit highly reliable timing (Puff-on: R2=1, slope=1.0000328­+8e-07, t(8)= 1.27e+06, p<0.001; Puff-off: R2=1, slope=1.0000335+6e-07, t(8)=1.57e+06, p<0.001; Light-on: R2=1, slope=1.0000334+6e-07, t(8)=1.57e+06, p<0.001; Light-off: R2=1, slope=1.0000315+7e-07, t(8)= 1.49e+06, p<0.001).

**Tables**

Table . Specialty components necessary to build a tone/light-puff system.

|  |  |  |  |
| --- | --- | --- | --- |
| **Part name** | **Website** | **Part number** | **Cost per unit** |
| Teensy 3.2 | https://www.pjrc.com/store/teensy32.html | TEENSY32 | $19.80 |
| 14x1 Double insulator pins (x3) | https://www.pjrc.com/store/header\_14x1\_d.html | HEADER\_14x1\_D | $0.85 |
| Prop shield | https://www.pjrc.com/store/prop\_shield.html | PROP\_SHIELD | $19.50 |

Table Specialty components necessary to build a motor output system

|  |  |  |  |
| --- | --- | --- | --- |
| **Part name** | **Website** | **Part number** | **Cost per unit** |
| Teensy 3.2 | https://www.pjrc.com/store/teensy32.html | TEENSY32 | $19.80 |
| ADNS-9800 sensors (x2) | https://www.tindie.com/products/jkicklighter/adns-9800-laser-motion-sensor/ | None | $27.50 |

Table General components necessary for work with both setups

|  |
| --- |
| **Miscellaneous parts** |
| SMA connectors |
| SMA coaxial cables |
| Solid wire (22 gauge used) |
| Soldering iron |
| Solder |
| Crimping tool |
| Wire stripper |
| Dupont connectors and housing |
| Male and female pin headers |