A low-cost environment for imaging neurons in various experimental settings

Introduction

Calcium imaging is a burgeoning technique used for imaging and assessing the collective activity of hundreds of neurons simultaneously. A new challenge is the development of techniques that allow for concomitant execution of different tasks and experimental paradigms along with calcium 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 operant conditioning.

An ideal experimental setup requires several components. First and most importantly, it must have high temporal fidelity. Calcium imaging has strict temporal requirements due to the fact that GCaMP must first be exposed to an LED for a fixed amount of time before an image is captured. A common digital design is to set up a CMOS or other imaging device to capture a frame every time it receives 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. Also, in order to train a mouse 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 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.

Finally, the experimental setup should be both widely accessible and open-source. These requirements have several sub-components that go hand-in-hand. First, it should be affordable, for reasons that are obvious. Current environments and programming environments can be exceedingly expensive. The Teensy 3.2 itself costs only $19.80 (<https://www.pjrc.com/store/teensy32.html>). The most expensive experimental component that we use in our setup is the ADNS-9800 sensor, which costs only $27.50. (<https://www.tindie.com/products/jkicklighter/adns-9800-laser-motion-sensor/>). The Arduino and Teensyduino programming environments are free, with the option of leaving a donation for continued development. Wide 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 they do more and more, we will see the development of new open source libraries accelerate. Cost can be prohibitive without grant money; 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.

Here, we introduce two specific implementations of calcium imaging experimental designs, implemented via a Teensy 3.2 microcontroller in conjunction with several simple code scripts, and thereby demonstrate the ease and usefulness of adopting such a design for future experiments.

**Methods**

**Motion tracking using the ADNS-9800**

Here we introduce a system for simultaneous wide-field calcium imaging and simultaneous motion three-dimensional treadmill tracking that necessitates only a Teensy 3.2 microcontroller (~$20.00), and two ADNS-9800 laser motion sensors (~$27.00x2) (<https://www.tindie.com/products/jkicklighter/adns-9800-laser-motion-sensor/>). This system offers an affordable, modular, open-source method of tracking mouse movement with high fidelity, temporal accuracy and without introducing confounding experimental variables.

*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. First, it has a higher clock rate than the Arduino (72 MHz vs 16 MHz), allowing for faster and 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, 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. This allows us to reliably acquire velocity estimates from our sensors at 20 Hz or at any other reasonable imaging speed.

*ADNS-9800 Sensors*

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>). We have included in our software package drivers for these sensors that allow for easy interfacing and reading from the “motion burst” register, which returns displacement in the x and y directions in precalibrated metric units. Further, accumulated displacements can be stored in the sensors between readings and digital pulses (which occur at nearly simultaneous time points). This is possible because ADNS-9800 sensors store motion data in 16 bits instead of the standard 8 bits.

*Experimental design*

The overall design for this experiment is shown in Figure [INSERT FIGURE NUMBER HERE]. Two ADNS-9800 sensors are attached at the equator of a container in which a large, buoyant Styrofoam ball is floating. These sensors lie at an angle of approximately 90 degrees from one another. To compute linear velocity, we can simply take the Euclidean distance of 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, the details of which can be seen in Figure [INSERT FIGURE NAME HERE]. This design can be achieved with inexpensive jumper wires and minimal soldering.

In order to begin experiments with the Teensy, we wrote a simple graphical user interfaces that can be used on a desktop or laptop. 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 the CMOS camera to capture a TIFF image, and also the frequency with which accumulated motor information will be recorded by this PC. The PC or laptop sends this information over a serial connection to the Teensy utilizing a bidirectional microUSB-USB cable.

In a proof-of-design experiment (Figure 3), we recorded an approximately 10 minute long session of a mouse running on a 3 dimensional treadmill at 20 Hz. Sensors were placed in a 3D-printed half-sphere at an angle of approximately 75 degrees from one another. 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). Times and distances travelled were recorded by the Teensy 3.2, and the timings of digital pulses were measured by an external device . A MATLAB graphical user interface was designed and used to initiate the 10 minute long experiment.

**Operant conditioning experiment**

To illustrate another simple experimental design wherein the Teensy 3.2 can be used to control multiple different devices and output a sound, we created a tone-puff setup. The corresponding setup is shown in Figure [X]. One 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 allows for many experimental additions, particularly the addition of sound. In fact, Teensy provides a 44.1 kHz audio library, through which a user can play or synthesize particular sounds. In our design, we output a simple 9500 Hz tone that a mouse will learn to associate with a “puff” that arrives following the sound.

In order to amplify the sound, we added a prop shield to this design. The prop shield is an affordable ($19.50, <https://www.pjrc.com/store/prop_shield.html>) add-on that is capable of amplifying the analog signal. A less expensive option is also available for $8.40 (<https://www.pjrc.com/store/prop_shield_lowcost.html>), as is 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 double insulator pins (<https://www.pjrc.com/store/header_14x1_d.html>), and then fed the output to a [X]-ohm speaker. 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 with widths of [X] ms were also constantly sent to a CMOS camera, triggering image captures every 50 ms. Again the “IntervalTimer” library was used in order to reliably orchestrate TTL pulses and the multiple output devices. The speaker, CMOS camera, puff, and light source were attached to the microcontroller using simple coaxial cables with SMA connectors.

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. Using this, a user can enter the length of the each trial and the frequency of the digital pulse output to the CMOS camera. The PC or laptop again 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 recorded by an external device.

**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 recording images via a CMOS camera via brief digital pulses. In the second (Figure 2Ai and 2Aii), we constructed a device capable of running a simple operant 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.

***Motor acquisition***

There are a number of ways in which people have attempted to observe motor output while imaging from the striatum. 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, no matter how light, 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 et al. (2007) and utilized widely elsewhere (Aronov and Tank, 2014; Gritton et al. (2018) (in review). 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 et al. (2007). Therefore, the mouse 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 (Dombeck et al., 2007, Aronov and Tank, 2014), which, though a comprehensive piece of software, is expensive and proprietary. In our own lab, implementing high-level MATLAB implementations of TTL pulse-based data acquisition using a National Instruments data acquisition board in conjunction with ViRMEN software led in certain to temporal delays and jitter. As described above, imaging needs a platform that is low-cost, scalable, and had high temporal fidelity. We have implemented such a system.

In our experiment, we replaced the more commonly used computer mice with gaming mouse sensors, the ADNS-9800, which are sold independently from the mice. In order to provide the end user with as simple a setup as possible, we designed drivers and a library that users can apply 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. Therefore, no calibrating is needed. 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 et al., 2007). Conversions to imperial or metric distances are implemented in the ADNS-9800 driver library that we have designed. Therefore, little must be done besides implementing the proper wiring in order to get such a design up and running that is capable of recording accurate x, y, and rotational displacements.

This design has high spatial and temporal fidelity. For example, Figure 3B demonstrates that the computer mice closely track the ideal

Our second design

**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 operant 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 operant-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.**

**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 + 5e-09 (t(11997)= 2.0381e+08, p < 0.001, R2=1), indicating an excellent fit and approximately 1:1 correspondence of time stamps. **C.** The difference of between times of digital pulses as measured by the external device, minus the times of the digital pulses as measured by the Teensy. The timing measured by the Teensy has high fidelity to the timing measured by the external device, differing in timing by approximately 29 microseconds per sample (linear model, difference vs time as measured by Teensy, R2=1, slope= 2.8927e-05 + 5e-09 (t(11997)=5800.1, p<0.001).

**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 + 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+8e-07, t(8)= 1.27e+06, p<0.001; Puff-off: R2=1, slope=1+6e-07, t(8)=1.57e+06, p<0.001; Light-on: R2=1, slope=1+6e-07, t(8)=1.57e+06, p<0.001; Light-off: R2=1, slope=1+7e-07, t(8)= 1.49e+06, p<0.001).