A flexible Teensy microcontroller based interface for wide-field optical imaging during behavioral experiments

Michael Romano, Mark Bucklin, Dev Mehrotra, Robb Kessel, Howard Gritton, Xue Han

**Abstract**

Systems neuroscience experiments often require the integration of precisely timed data acquisition and behavioral control. While many commercial systems have been designed to meet various needs, they often fail to offer flexibility that allows for integration across diverse or new experimental designs. . For example, it has been difficult to integrate recently developed sCMOS cameras with various input and output devices, for high speed, large scale calcium imaging analysis of neuronal circuits during behavior. We here developed Teensy 3.2 microcontroller-based interface that offers high-speed, precisely timed digital data acquisition of behavioral data, and digital and analog outputs for controlling sCMOS cameras and other devices. We demonstrate the efficacy and the temporal precision of the Teensy interface in two experimental settings. In one example, we demonstrate the use of Teensy interface for reliable recording of an animal’s directional movement on a spherical treadmill, along with simultaneous control of an imaging device for high speed integrated image acquisition. In another example, we demonstrate the use of Teensy interface for temporally precise delivery of auditory and visual signals in a trace conditioning learning behavioral paradigm, while controlling a camera for image acquisition. These examples demonstrate that Teensy 3.2 equipped with its hardware modules provides an efficient and flexible platform capable of integrating imaging devices synchronously into behavioral experimental designs, for high-speed and temporally precise imaging and behavior output for systems neuroscience experiments.

**Introduction**

Recent advances in sCMOS cameras and genetically encoded calcium sensors enable neuroscientists to perform fluorescence imaging of thousands of individual cells’ activity, allowing the analysis of networks related to behavior (Klaus, et al., 2017; Barbera, et al., 2016; Mohammed, et al., 2016; Markowitz, et al., 2018). However, the integration of high speed sCMOS cameras with devices needed to monitor and control behavioral progress has been difficult. In particular, it has been challenging to simply design different experiments with precise and consistent imaging rates while synchronizing experimental event timing with frame capture.

Experiments that examine the neural basis of behavior typically require precisely timed data acquisition and command signals, as noted previously for electrophysiology recordings (Solari, Sviatkó, Laszlovsky, Hegedüs, & Hangya, 2018). There are many strategies to control the temporal precision of an experiment, where neural activities can be precisely aligned with behavioral progress, such as various Analog/Digital interface that can be precisely controlled by LabVIEW and Matlab. However, depending on how the experiment is designed with these programs, it can lead to variability in experimental timing, including frame capture, data acquisition, or stimulation, and necessitate interpolation or other post-hoc methods to align imaging data with motor data. This is due to the fact that these computer programs utilize a full operating system that must balance multiple operating system processes at once. One can potentially design an experiment with either one of these environments that operates in a highly precise manner with low variance, but this can be more challenging. New microcontrollers offer this capability, however, with a very low learning curve for programming.

Over the last several years, microcontrollers traditionally used by hobbyists have gained popularity across a variety of scientific fields (Sanders & Kepecs, 2014; D'Ausilio, 2012; Chen & Li, 2017; Husain, Hadad, & Zainal Alam, 2016). Microcontrollers are small, open-source, and low-cost, allowing for easy customization and implementation. Arduinos were one of the first major microcontroller to gain substantial popularity. They are capable of delivering precisely timed digital outputs with microsecond time resolution, while using user-friendly software functions. Recently, the Teensy 3.2 was developed, which has the feature of delivering analog output in addition to the features of the current version of the Arduino UNO, for example. Teensy uses the same software environment as Arduino, which is intuitive to learn without the need of advanced programming experience (D'Ausilio, 2012). Because these microcontrollers are low cost, they can be easily scaled for multiple experiments simultaneously. Since the software is open-source and the programming language is intuitive (D'Ausilio, 2012), microcontrollers can be easily adapted to various experimental needs including the integration of newly developed instruments.

Camera control via an Arduino device that initiates the start of an imaging sequence has been previously shown (Micallef, Takahashi, Larkum, & Palmer, 2017). However, a limitation of this approach is that it is necessary to synchronize frame timing with behavioral data after the experiment is complete, which is inexact and likely necessitates interpolation. Arduino and Teensy devices can instead be used to precisely time imaging capture for each frame. A common technique in laboratory studies using more expensive AD converters is to set up an imaging device to utilize an “external trigger” (Micallef, Takahashi, Larkum, & Palmer, 2017), where the rising phase of a digital pulse or TTL pulse either initiates a sequence of internally clocked image captures or initiates each individual image capture. One possible concern with the latter approach is that imprecise triggering of each frame based on a different digital pulses can introduce jitter in digital pulse delivery, potentially causing frame loss and potentially necessitating interpolation for many statistical analyses. Thus, there currently exists a need to engineer a device capable of delivering continuous, precisely timed digital pulses that can be delivered synchronously with other experimental events with camera control.

Here, we demonstrate in two simple experimental paradigms that highly accurate data acquisition, and sound and stimulus delivery synchronized with image capture is simple and achievable via a Teensy 3.2 and the two corresponding software implementations. The Teensy 3.2 is capable of keeping highly accurate and low-bias timing that allow it to reliably instantiate frame capture with regular intervals while delivering stimuli or recording experimental data with microsecond-level precision, which makes this an ideal design for frame-by-frame control and recording of imaging experiments. Further, this microcontroller offers analog output and easy-to-program environment, making it highly flexible and worthy of widespread utilization.

**Methods**

*Motor acquisition experiment*

In this experiment, we performed motion tracking using two ADNS-9800 gaming sensors (<https://www.tindie.com/products/jkicklighter/adns-9800-laser-motion-sensor/>, Tindie, “[NS-9800 Laser Motion Sensor](https://www.tindie.com/products/jkicklighter/adns-9800-laser-motion-sensor/)”, see Table 1), while delivering digital pulses that could be used to trigger a camera, for example an sCMOS camera, at 20Hz. The overall design for this experiment is shown in Figure 1A. Two ADNS-9800 gaming sensor boards were attached at the equator of a 3D-printed half-sphere in which a large, buoyant Styrofoam ball is floated by house air as described previously (Dombeck, Khabbaz, Collman, Adelman, & Tank, 2007). These sensors lay at an angle of approximately 75 degrees from one another, so that the y-readings of both sensors can be used to compute linear velocity, and the x-readings can be used to compute rotational velocity.

These two sensors were connected to a Teensy 3.2 (PJRC, TEENSY32) via simple serial peripheral interface (SPI) connections with insulated 22 gauge wires i.e. Elenco SolidHook-up Wire (<https://www.amazon.com/Elenco-Hook-Up-Colors-dispenser-WK-106/dp/B008L3QJAS/>, Amazon, ASIN=B008L3QJAS), as shown in Figure 2A. We utilized a crimping tool (<https://www.amazon.com/IWISS-Professional-Compression-Ratcheting-Wire-electrode/dp/B00OMM4YUY/>, Amazon, ASIN= B00OMM4YUY) to attach crimp pins and housing to the ends of the wires (for example, <https://www.amazon.com/gp/product/B0774NMT1S/>, Amazon, ASIN= B0774NMT1S) in order to connect them to the Teensy and to the sensors. Output from the Teensy was directed from pins to SMA connectors via 22 gauge wires, and from SMA connectors toward external devices via SMA cables (for example, <https://www.amazon.com/Uxcell-a11053100ux0317-Connector-Straight-Adapter/dp/B006Z95OEC/>, Amazon, ASIN= B006Z95OEC), and to a PC using a USB-microUSB cable.

The output from Teensy, representing frame capture triggers, was measured by an external device at 3051.76 Hz (Tucker-Davis Technologies RZ5D (TDT RZ5D)). To acquire motor sensor data and to send digital pulses, we utilized the “IntervalTimer” function available in the standard Teensy library, which allows for calling different functions with microsecond precision. We used it to call a main function that sends out a digital pulse to capture a frame, collects data from the two ADNS-9800 sensors, and then sends the motion data to a computer.

To extract readings from these sensors, we utilized functions that are freely available on Github (<https://github.com/markbucklin/NavigationSensor>), which contain a modified version of the ADNS-9800 library (<https://github.com/mrjohnk/ADNS-9800>). Via this modified ADNS-9800 library, we read from the “motion burst” register of each sensor. On every call to the main function, we acquired the accumulated displacement over the previous 50 milliseconds in both the x and y directions. For the counts per inch setting we used a value of 3400 counts per inch, the default setting. As previously mentioned, during this interrupt, a digital “on” pulse that lasts for 1 ms is sent out of a digital pin using the DigitalIO library (<https://github.com/greiman/DigitalIO>). This library 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” (to INPUT or OUTPUT, for example). 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.

To set the sampling frequency and length of the experiment in the Teensy and trigger the beginning of an experiment, after the main script was uploaded to the Teensy, 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, however, this graphical user interface could be written in Python or any other programming language. Using this interface, the user enters 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 an sCMOS camera to capture an image, 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 a bidirectional microUSB-USB cable.

In a proof-of-concept experiment (Figure 3), we recorded a 10 minute long session of a mouse running on the spherical treadmill, and data was acquired at 20 Hz, with concomitant digital outputs that could be used to trigger an sCMOS camera for image capture or a different device. The mouse’s speed was computed using the y-coordinates of each ADNS-9800 sensor, and the total distance travelled at any one time point was computed using the following equation:

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 (50 ms)s. Times and distances travelled were recorded by the Teensy 3.2, and the timings of digital pulses

After analyzing the time stamps acquired by the TDT RZ5D system, we noticed that there was a very small timing drift (approximately 30 microseconds per second). To confirm that the frequency of data acquisition and timing of the corresponding digital pulses didn’t affect this drift, we repeated 5 minute recording sessions without a live mouse at 20, 50, and 100 Hz. These recordings used an identical script, except we embedded a 500 microsecond delay between the start and end of the digital pulse (“delayMicroseconds(500)”) instead of a 1 millisecond delay (“delay(1)”).

*Classical conditioning experiment*

In this experiment, we utilized the Teensy to deliver a sound and puff to control the progress of a trace conditioning behavioral paradigm, while delivering digital pulses to an sCMOS camera that could be used to trigger image acquisition at 20Hz. The general setup is shown in Figure 1B, and the specialty components required to build this design are shown in Table 2. As with the motor control setup, single stranded 22 gauge wires were used to direct output from each of the utilized Teensy pins to SMA adapters, from which SMA cables were used to send output to the desired devices. In a trace conditioning experiment utilizing this setup, a head-fixed mouse would first be exposed to a 9500 Hz tone concomitantly with a light stimulus, and then receive a gentle eye puff after a brief memory trace interval.

To deliver an audible sound through Teensy, we used a prop shield module available for Teensy (PJRC, PROP\_SHIELD), which is an easy-to-use add-on capable of amplifying analog output signals (shown in Figure 2B as pin A14). If stereo outputs are desired, the manufacturer also offers an audio shield (<https://www.pjrc.com/store/teensy3_audio.html>) that is capable of stereo output, as demonstrated previously (Solari, Sviatkó, Laszlovsky, Hegedüs, & Hangya, 2018). Prop shield was attached to Teensy with 14x1 double insulator pins (PJRC, HEADER\_14x1\_D), with the output connected 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 such that they could activate a LED light concomitant with the sound, and a puff as an aversive stimulus following each sound/light combination. Meanwhile, digital pulses to control sCMOS camera were programmed to occur every 50ms.

We here utilized “elapsedMicros” in order to reliably time all of the experimental events. “elapsedMicros” objects serve as time incrementers, that increment time at the microsecond time scale restarting every time that its value is set to zero. Every 50 ms, this code called a main function that updated the status of the digital pins associated with the “puff” and the light stimulus, and updated the amplitude of the 9500 Hz sine wave (amplitudes were set to 0.05 during audio stimulus time periods, and 0 elsewhere). Also, at the termination of a trial, this function incremented the trial number. Finally, a 1 ms digital pulse was delivered via another pin to instantiate a theoretical camera trigger. 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.

In order to begin experiments with the Teensy, we wrote in MATLAB a simple graphical user interface that can be used on a desktop or laptop. With this, a user enters the length of the each trial and the number of trials desired. 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, and the experimental and trial times (in milliseconds) at the beginning of each IntervalTimer function call.

In our proof-of-concept experiment (Figure 3), the puff, tone, and camera trigger pins were all attached to and were recorded by the same external device (TDT RZ5D) at 3051.76 Hz for the puff and camera trigger pins, and 24414.0625 Hz for the tone pin. The tone pin was measured directly (not through the amplifier). We performed a mock-recording consisting of 50 trials of 20 seconds length each, where sound and light output pins were programmed to turned on 11.1 seconds into each trial for 700 ms, and the pin used to generate the aversive puff stimulus was turned at 12.05 seconds into each trial for 100 ms.

In order to measure latency (Figure 4Bi and iii), we acquired the timing of the camera digital pulse, according to the TDT system, that corresponds to the exact frame start at approximately which point the audio signal was turned on. We then acquired the timing of either the puff pin onset or the timing of the onset of the audio signal. In order to measure the onset of the audio signal, we took the raw recording and high-pass filtered the signal using a 6th-order Butterworth filter, a bandpass frequency of 1000 Hz, and a “zero-phase digital filter” (MATLAB command “filtfilt”). Then, we took the absolute value of the Hilbert transform of the filtered signal to acquire an amplitude envelope. After finding the amplitude envelope, we found those values that exceeded a value of 0.005. The first time point that the amplitude crossed this threshold was considered the tone onset, and the next time point that dropped below this threshold was considered the tone termination.

*Statistics*

Linear models were constructed using the “fitlm” function in MATLAB 2017b. Theoretical timings, to which measured timings were compared, were each taken to be timings beginning at 0 seconds in equal increments of 50 milliseconds for both experiments.

**Results**

A widely utilized design in imaging experiments utilizes commercial data acquisition boards in conjunction with software written and executed by a PC. However, PCs run a multitude of processes simultaneously, and it is therefore challenging to reliably time experimental events. An even more challenging task is aligning camera frames with experimental events following an experiment. For example, initiating only the beginning of a recording session or trial with a PC and then having imaging and behavioral control and acquisition run separately poses two problems. First, if one uses the PC to generate digital pulses via a data acquisition board to initiate frame capture for every frame individually at some fixed frequency in conjunction with dictating behavioral event control or data acquisition, it is likely that the actual rate of frame capture will exhibit some variability. As previously noted, this is due to the fact that PCs have to balance the execution of multiple tasks simultaneously. However, with this design, an experimenter would still be able to programmatically synchronize experimental events with specific frames, as everything will run based on the same clock (the PC’s clock). Second, an alternative design involves using a PC only to initiate the beginning of a trial, with the camera then imaging at a fixed frequency independently of the behavioral aspects of the experiment. This solves the issue of camera jitter, but introduces two other issues. First, behavioral events cannot be measured with respect to frame capture, necessitating some kind of interpolation to align behavioral and imaging data. Second, relative timings between the camera and the behavioral data acquisition could experience timing drift if their clocks run at slightly different rates. In this case, even interpolating data won’t fully recover accurate alignment between the two data sets. Thus, an ideal experimental design has one clock kept by a single experimental controller that provides precise timing of image capture while simultaneously controlling and monitoring behavioral data synchronously with the image capture. Microcontrollers help to fill this need.

Microcontrollers such as Arduino UNOs have gained popularity in neuroscience research due to their user-friendly interface and their flexibility of device integration (D'Ausilio, 2012; Chen & Li, 2017; Micallef, Takahashi, Larkum, & Palmer, 2017). However, the Arduino UNO does not have direct analog output. Recently, the Teensy 3.2 (<https://www.pjrc.com/store/teensy32.html>) has been developed, which not only has analog output and a comprehensive Audio library, but also has the capability to use the IntervalTimer function. This function takes as input a single main function and the time, in microseconds, desired between calls to this function. It is highly accurate and is particularly well suited for experiments that require precise, repeated executions of a particular task. In addition, the Teensy 3.2 software has the built-in capability to utilize the elapsedMicros and elapsedMillis libraries. These libraries serve as highly accurate time accumulators that can be used to time experimental events to microsecond or millisecond accuracy, respectively. This is a desirable alternative to the IntervalTimer when the “interrupts” utilized by the IntervalTimer could interfere with other components of the code, such as audio output. Here, we present Teensy based interface to integrate sCMOS camera image acquisition and behavioral experimental control.

*Motion tracking using ADNS-9800 sensors*

To demonstrate the feasibility of a Teensy-based interface for synchronous data acquisition and camera control during behavioral experiments, we constructed a setup (Figure 1Ai and 1Aii), to record animal locomotion data from two ADNS-9800 motion sensor boards reading the position of a spherical treadmill, while delivering digital outputs to a camera. As shown in Table 1, the total cost is approximately $80.

To measure locomotion from awake head fixed mice, we utilized a Teensy interface to record from two ADNS-9800 motion sensors. These sensors are affixed to a “spherical treadmill” setup, as described by Dombeck, Khabbaz, Collman, Adelman, & Tank (2007). Mice were surgically fitted with a head plate and imaging window, and head-fixed above a house air floated ball (Figure 1Aii).

ADNS-9800 sensor boards are inexpensive, and are more sensitive than regular computer mice as used in previous designs and can measure up to 8200 counts per inch, providing a more accurate measure of locomotion parameters. Additionally, ADNS-9800 sensors have a high maximum sampling rate of 12000 frames per second, so multiple readings per image capture are possible (<https://datasheet.octopart.com/ADNS-9800-Avago-datasheet-10666463.pdf>). For example, if imaging at 20 Hz, one could design a script to record motor data every 0.1 ms, and synchronize camera capture to every 500th frame. This would give an even more precise account of motor information while maintaining camera-frame alignment. Further, accumulated displacements can be stored in the sensors between readings, because ADNS-9800 sensors store motion data in 16 bits instead of the more standard 8 bits. Therefore, sensor saturation is not a concern at moderate sampling rates. That is, despite the much higher sensitivity of these sensors, it is not necessary to query them for motion information any more frequently than a standard sensor.

These ADNS-9800 sensors were controlled via the ADNS9800 library, found freely at <https://github.com/markbucklin/NavigationSensor/src/ADNS9800>. With these sensors, we read displacements and converted them directly to micrometer displacements using their internal calibration. Proper wiring is simple and is demonstrated in Figure 2B. The connections demonstrated using dotted lines can be replaced with jumper wires or sturdier, longer lasting wire, as detailed in the *Methods*. No external capacitors or resistors are needed.

To test the fidelity and temporal accuracy while maintaining alignment with imaging data, we recorded the movement of a mouse while it was running on the spherical ball. We calculated the velocity of the mouse, with an average of 7.1 + 6.9 cm/s, with a maximum velocity of 47.0 cm/s, which is in agreement with the general observation as with previous studies (see, for example, (Dombeck, Khabbaz, Collman, Adelman, & Tank, 2007)). When we measured the digital output timing with the TDT RZ5D, we found that digital outputs are precise, with a near-perfect linear relationship, and a 28.9us per second positive bias, thus the actual frequency was 19.999 Hz instead of 20 Hz. The linear model fit demonstrated a root mean squared error of 38.9 microseconds. This indicates that the camera trigger has at least microsecond-level precision.

To further verify the accuracy of the IntervalTimer, we repeated recordings at 20Hz, 50Hz and 100Hz, and found that the actual frequencies were 19.999, 49.999, and 99.997 Hz, respectively. This accuracy is consistent with that previously reported using an Arduino UNO: with repeated sampling of single 900 ms long TTL pulses with 100 ms inter-pulse intervals, the average length of time between sequential pulses was 1000.6 milliseconds (D'Ausilio, 2012). Together, these results demonstrate the temporal precision and accuracy of the Teensy in conjunction with the IntervalTimer function. In addition, it underscores the utility of the Teensy in triggering synchronous frame-capture during long recording experiments for precise alignment of neuronal data with behavioral states (Micallef, Takahashi, Larkum, & Palmer, 2017).

*Trace conditioning*

In the second experiment (Figure 1B and 2B), we constructed a Teensy-based setup for trace conditioning learning experiment, where a mouse is trained to associate conditioned stimuli (tone and/or light) with a subsequent unconditioned stimulus (an eye puff) separated by a brief time window. We set up the Teensy to deliver conditioned stimuli and to record the timing of each of these events, whose state changes were synchronized to frame capture. To deliver an auditory stimulus, we used a plug and play hardware amplifier (prop shield) to amplify the analog output from the Teensy, which can then drive speakers of both 4 and 8 ohms. Three sets of 14x1 double insulated pins for connecting the Teensy to the prop shield. In total, this setup costs approximately $40, excluding general equipment.

We recorded the timings of each digital output and compared them to the theoretical timings with samples spaced at exactly 50ms apart, as shown in Figure 4A. Similar to the observation in the locomotion experimental design, the measured timings were close to the theoretical timings, with a 33.4 microsecond delay per sample. Notably, concomitant execution of audio and puff pins did not appear to greatly alter either the slight timing drift or precision of the digital pulses directed at the sCMOS camera compared with the motor setup, which experienced a similar delay per sample (28.9 us). The root mean squared error for the model fit was 13.3 us, consistent with the motor setup in its microsecond-level precision.

We further quantified sound onset latency, sound duration, puff latency, and puff duration. As shown in figure 4Biii, the digital output for eye puff showed no delay from the theoretical time (mean latency = -0.004 + 0.012 ms, range=0.04 ms seconds). The duration of the puff digital pulse was also both highly accurate and consistent, and lasted 100.03+0.02 (+ std) ms over the course of the 50 trials, hardly differing from the expected duration of precisely 100ms.

Sound latency was measured by the difference between the timing of the digital pulse delivered sequentially with the tone amplitude change, and the time of the tone amplitude change as measured by the TDT system. While non-negligible, tone timing was both precise and predictable: it averaged 7.6 + 0.9 milliseconds. Because of the consistency of the timing latency, it is easy to adjust for this latency within the code, in this case by instantiating a change in signal amplitude 7.6 milliseconds earlier than the corresponding frame capture. The value that we observed was similar to the value of 6.9 + 0.9 milliseconds that was observed in a similar design utilizing a Teensy 3.2, where the Teensy was used to play a pre-recorded sound after stimulation by a Bpod behavioral control system (Solari, Sviatkó, Laszlovsky, Hegedüs, & Hangya, 2018).

We also measured the lengths of the tone, which was highly accurate. It lasted for 700 + 1 ms, with a range of 2.9 ms. Other implementations of the Audio library could potentially offer even more precision, if so desired. For example, if one needed to utilize a precise sound sequence in an experiment, they could upload the sound sequence as a .wav file and utilize the Teensy to play the pre-recorded sound (Solari, Sviatkó, Laszlovsky, Hegedüs, & Hangya, 2018). However, our design can be implemented very simply, utilizing additional code only within the main startup script. Ultimately, the precisions of both our puff and sound output are comparable to expensive, available systems such as the Habitest Modular system in conjunction with Coulbourn Graphic State 4 software, which itself offers 1 ms precision (<http://www.coulbourn.com/v/vspfiles/assets/manuals/Graphic%20State%204%20Users%20Manual.pdf>) , making the Teensy a viable, inexpensive alternative that is also able to capture synchronous imaging data using our software design.

**Conclusion and Discussion**

We demonstrate the use of Teensy 3.2 microcontroller in integrating synchronous sCMOS camera image capture with various devices for behavioral experiments. In one novel experimental design, we utilized recently developed ADNS-9800 gaming sensors for precise and high speed locomotion tracking, along with simultaneous camera commands. In a second experiment, we commanded four devices with precise timing during a trace conditioning experiment. In both experiments, the timing of the Teensy interface was accurate to within approximately 30 us, and precise, as measured by the root mean square error of the model fits, to the level of microseconds. We developed two user-friendly software interfaces, experimental scripts and simple hardware designs for both experiments. Together, these software and hardware configurations can be immediately adopted for the designed behavioral experiments, or customized for other types of behavioral experiments, where camera-based imaging is desired during behavior. This platform is additionally low cost and can be easily scaled for parallel experiments across many animals, or further customized for various types of behavioral experiments.

We also demonstrate a setup built to implement a trace conditioning paradigm. In addition to requiring accurate alignment of imaging with behavior, operant conditioning paradigms need reliable stimulus timing. In this setting, repetition of stimulus and response must occur in a highly regular temporal fashion in order for a mouse to learn and in order for the neuronal response to be consistent. This illustrates the ability of the Teensy to orchestrate different classes of output—analog and digital, both long and short pulses—simultaneously and with high temporal accuracy while simultaneously and synchronously sending out regular digital pulses to control an image capturing device. It also highlights the ability of this device to simultaneously produce an analog output, in particular to generate a sound, while performing other actions. As previously stated, a major advantage of the Teensy 3.2 over other microcontrollers such as the Arduino UNO is the fact that it can output a true analog signal. 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-like signal. Rather, the Teensy 3.2 simply needs to be soldered on to a paired hardware module (prop shield), and less in-depth knowledge about electronic circuits is necessary. In addition, it has a built-in “Audio” library that simplifies sound synthesis, reading, and mixing, all at 44.1 kHz, which is stereo quality.

An important discovery during development of this system is that realization of a slight linear drift of the Teensy processing clock. This drift is linear in nature, which makes it simple to calibrate out if actual sub-µs precision to real world timing is essential. Further, it actually underscores the necessity of a central controller for precise acquisition and total experimental control. Synchronizing different devices only by a single pulse at the start of an experiment can lead to problems when trying to acquire motor output or deliver some experimental stimulus and examine cellular behavior with high temporal accuracy. Initiating experimental events from a high-level source, such as directly from a PC, can introduce timing jitter due to the multitude of tasks that a PC must attend to at any given point in time. For example, a recent calcium imaging study in the striatum finds additional neurological structure related to motor activity on very short timescales, but finds only velocity correlated with neural activity on longer timescales (Markowitz, et al., 2018). This suggests that with sufficient timing jitter, correlations on short time scales could be missed, yielding the conclusion for example that velocity is the only correlate of neural activity in the striatum.

Further, with concomitant imaging, one must also align tasks to imaging data after the fact, or else face substantial variability in frame spacing. As explained previously (D'Ausilio, 2012; Solari, Sviatkó, Laszlovsky, Hegedüs, & Hangya, 2018), using a microcontroller such as an Arduino or the Teensy 3.2 circumvents the issue of imprecise timing of behavioral events. We note that in addition, synchronizing camera triggers with experimental events circumvents the need of post-hoc image alignment.

In conclusion, the Teensy 3.2 enables a user to flexibly orchestrate experiments with synchronous behavioral monitoring control and capture with image capture. Additional timing functions, such as the “IntervalTimer”, make the Teensy 3.2 easier to program to deliver equally spaced, regular digital pulses needed for triggering image capture. Further, the Teensy 3.2 allows a user to generate analog signals. The precision and flexibility of the Teensy microcontroller make this a user-friendly, easily adaptable, accurate, and precise tool for to utilize in different experimental designs that benefit from synchronous image capture and behavioral control and acquisition.

**Figures**

**Figure 1.** Diagrams of the two experimental device setups using Teensy interface. A, a floating, spherical treadmill setup for locomotion recording (A), **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** A tone/light and puff trace conditioning setup. 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. 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.** Detailed electrical schematics for both the motor-control and tone-puff systems **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.** Example recording using the motor control setup **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**. **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. Green indicates linear model, and in black are experimental data, down-sampled by a factor of 200. The linear model estimates a slope of 1.000028927 + .000000005 (t(11997)= 2.0381e+08, p < 0.001, R2=1; intercept = 0.000107 + 0.000002, t(11997) = 63.243, p < 0.001), indicating an excellent fit and very nearly a 1:1 correspondence of time stamps.

**Figure 4.** Example recording using the tone-puff setup**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: 1.0000334 + 0 (to machine precision), t(14998)=infinite, p<0.001). **B.** Timing measured by the teensy for (i) and by the TDT system for (ii-iv) over the course of fifty trials; (i) shows the latency between the theoretical onset of the tone and the measured timing of the tone as measured by the TDT device (mean=7.6 + 0.9 ms, range=2.9 ms); (ii) shows the consistency of the length of tone intervals across all trials (mean=700 + 1 ms, range=2.9 ms); (iii) shows the consistency of the latency of the puff stimulus, as measured by the TDT sytem (mean= -0.004 + 0.012 ms, range=0.04 ms); (iv) shows the consistency of the length of the puff across all trials (mean = 100.03+0.02 ms). (all + std).

**Tables**

Table 1. 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 | https://www.tindie.com/products/jkicklighter/adns-9800-laser-motion-sensor/ | ADNS-9800 Laser Motion Sensor | $27.50 |

Table 2. Specialty components necessary to build a tone-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 | 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 |

# References

Aranov, D., & Tank, D. W. (2014). Engagement of Neural Circuits Underlying 2D Spatial Navigation in a Rodent Virtual Reality System. *Neuron, 84*(2), 442-456.

Barbera, G., Liang, B., Zhang, L., Gerfen, C. R., Culurciello, E., Chen, R., . . . Lin, D.-T. (2016, October 5). Spatially Compact Neural Clusters in the Dorsal Striatum Encode Locomotion Relevant Information. *Neuron, 92*(1), 202-213.

Chen, X., & Li, H. (2017, December). ArControl: An Arduino-Based Comprehensive Behavioral Platform with Real-Time Performance. *Frontiers in Behavioral Neuroscience, 11*, 1-9.

D'Ausilio, A. (2012). Arduino: A Low-Cost Multipurpose Lab Equipment. *Behavior Research Methods, 44*(2), 305-313.

Dombeck, D., Khabbaz, A. N., Collman, F., Adelman, T. L., & Tank, D. W. (2007). Imaging Large-Scale Neural Activity with Cellular Resolution in Awake, Mobile Mice. *Neuron, 56*(1), 43-57.

Klaus, A., Martins, G. J., Paixao, V. B., Zhou, P., Paninski, L., & Costa, R. M. (2017). The Spatiotemporal Organization of the Striatum Encodes Action Space. *Neuron, 95*(5), 1171-1180.e7.

Markowitz, J. E., Gillis, W. F., Beron, C. C., Neufeld, S. Q., Robertson, K., Bhagat, N. D., . . . Datta, S. R. (2018). The Striatum Organizes 3D Behavior via Moment-to-Moment Action Selection. *Cell, 174*(1), 44-58.e17.

Micallef, A. H., Takahashi, N., Larkum, M. E., & Palmer, L. M. (2017, May). A Reward-Based Behavioral Platform to Measure Neural Activity during Head-Fixed Behavior. *Frontiers in Cellular Neuroscience*, 1-8.

Mohammed, A. I., Gritton, H. J., Tseng, H.-a., Bucklin, M. E., Yao, Z., & Han, X. (2016). An Integrative Approach for Analyzing Hundreds of Neurons in Task Performing Mice Using Wide-Field Calcium Imaging. *Scientific Reports, 6*, 20986.

Solari, N., Sviatkó, K., Laszlovsky, T., Hegedüs, P., & Hangya, B. (2018). Open Source Tools for Temporally Controlled Rodent Behavior Suitable for Electrophysiology and Optogenetic Manipulations. *Frontiers in Systems Neuroscience, 12*(May).

Yoav, A., Kim, J. J., Brinks, D., Lou, S., Wu, H., Mostajo-Radji, M. A., . . . Cohen, A. E. (2018). All-Optical Electrophysiology Reveals Brain-State Dependent Changes in Hippocampal Subthreshold Dynamics and Excitability. *bioRxiv*. doi:https://doi.org/10.1101/281618