Timecode Systems Neural/Video Data Synchronization Proof-of-Concept

Xavi Loinaz

Introduction

The goal of this project was to develop a proof of concept for an Arduino-based method to synchronize various types of neural experimental data using Timecode Systems' wireless synchronization technology. Timecode Systems' wireless synchronization technology is often used for applications such as recording events, such as weddings or sporting events, but it is not used as commonly in electrophysiological research settings to synchronize data taken from patients/test subjects.

In various studies at the Brown University Neuromotion Laboratory, neural data is taken from humans and primates while they are recorded using both cameras and microphones, as well as sensors for other behavioral data. Precise synchronization of these various recordings of data becomes necessary, as it becomes important to know when certain neural events correspond to certain video frames and other behavioral data. Without this synchronization, the data being recorded by the various devices can be prone to drift from one another. Different devices have their own internal clocks and they run at slightly different rates, so the timestamps from each of the devices may eventually not precisely correspond to one another as the devices' internal clocks drift from one another. Thus, we cannot precisely know what motions of the test subject recorded on certain video frames correspond to which neural patterns, since the timestamps of the respective devices may have drifted from one another.

There does not necessarily exist a universal way to synchronize these data recordings, and talking to graduate students from the Neuromotion Lab, there appeared to be two main ways to synchronize the data. Graduate students Marc Powell and Radu Darie seemed to have systems that synchronized according to the clock of a neural signal processor, and graduate student Nicole Provenza utilized the Timecode Systems synchronization devices to send a master pulse acting as a universal timecode to all her devices.

Marc developed a synchronization system where all data acquired is sampled through a neural signal processor (NSP), which in this case is the Blackrock Cerebus system. A transistor-transistor logic (TTL) signal corresponding to video frame capture for both the Flea3 camera and behavioral data rig is sent into the NSP indicating data capture along with the corresponding timestamps sent as serial encoded bit streams from the clocks of the Flea3 camera and behavioral data rig themselves. From that, we can find when various data capture happened relative to each other using the clock of the NSP. We can see when a TTL signal is received according to the clock of the NSP, and from that we can find the corresponding timestamp relative to the clock of the original data capture device, and from that we can find the corresponding data on the original data capture device, thus allowing us to see when certain data were captured all relative to the clock of the NSP. Additionally, Marc had an audio recording device which was only capable of receiving data rather than sending it. In order to synchronize with the behavioral data, the behavioral data rig can also send a TTL signal into the audio recording device, such that it can be known when the audio data is acquired relative to the behavioral data. Thus, because we know when the behavioral data occurs relative to everything else since its timestamps are sent into the NSP, we also know when the audio data happens relative to all other devices. A basic schematic of this setup is shown in Figure 1.

Radu had a similar setup to Marc, albeit with different cameras (Simi cameras). A problem that Radu had with his setup was that certain frames would be dropped in the recordings of his Simi cameras without known corresponding timestamps, so his stream of frames could be missing one or two frames and he would not be able to match these dropped frames to appropriate timestamps. (Timestamps would not be dropped, so the stream of frames could be shifted by a couple timestamps.)

Nicole, however, utilized Timecode Systems' synchronization devices to synchronize the acquisition of her data. Nicole used Timecode Systems' :pulse device to act as a master device which sends out a linear timecode (LTC) signal to slave devices, which in this case were the SyncBac Pro and the UltraSync ONE. The SyncBac Pro jams the timecodes sent from the :pulse into GoPro HERO6 cameras, and the UltraSync can jam the timecodes sent from the :pulse into various other data acquisition devices, such as a microphone and the Open Ephys board, which is used to collect neural data. Since all devices have the same LTC from the :pulse in their data streams at a given time, all their data can be synchronized and matched up time-wise. A basic schematic of this is shown in Figure 2.

Nicole, however, was having issues with her synchronization setup in that the SyncBac Pros did not seem able to properly connect to the GoPro cameras, and thus could not jam the timecodes from the :pulse into them. Thus, the GoPro cameras for Nicole had to be synchronized through an alternative, less convenient method, which in this case involved using audio.

In this project, we create a proof-of-concept for synchronizing various types of data acquisition in an electrophysiological recording context. An Arduino Uno is used to generate a mock neural signal and to read it. Normally, an Arduino would not have enough input pins to receive all the neural signals that are normally recorded in these types of studies, but this setup serves to demonstrate that such an analog signal can be synchronized time-wise with video.

Methodology

I had a few iterations with regard to the setup for the data synchronization system I wanted to create.

First, I thought it would be wise to develop a universal synchronization system similar to what Marc developed, except that all data would be sampled through an Arduino. The idea was that any type of behavioral and/or neural sensor could be attached to the Arduino and all data would be able to successfully synchronize with one another. A schematic of this is shown in Figure 3. After talking with Prof. Borton, however, I realized this would not be possible because the NSP that is incorporated into these types of studies has far greater neural recording channels than the Arduino has input pins. Thus, the timestamps of all the neural recordings taken by the NSP would not necessarily be able to be received by the Arduino at any given time.

The project pivoted to developing a proof-of-concept for data synchronization using Timecode Systems' synchronization equipment along with Arduinos. Arduinos, which are programmable microcontrollers, could be used to interface with sensors used to collect data in this proof-of-concept. They have also been commonly used for behavioral data collection in the Brown Neuromotion Laboratory. In this case, the Arduinos could be used to both generate mock neural/behavioral data as well as acquire the mock neural/behavioral data. We can test synchronization of the Arduino data collection rig with cameras by having an LED that flickers at known times on the Arduino. If there are the same universal timecodes received from the :pulse device that correspond to when the Arduino turns the LED on and when the cameras record the LED as being on, then it can demonstrate synchronization of data between the camera and mock neural/ behavioral data collector, thus acting as a proof-of-concept for Timecode Systems-based synchronization of data for such electrophysiological recording studies. The corresponding schematic for this proof-of-concept setup is shown in Figure 4.

However, when actually implementing the schematic fro Figure 4, I realized the setup for the mock neural data Arduino and the mock behavioral Arduino were identical, and thus redundant. It could have value in ensuring that the data acquisition between multiple Arduino rigs are synchronized, but I simplified things for now and only used one Arduino rig. Thus, I reduced my setup to be that shown in the schematic of Figure 5.

To carry out the setup shown in Figure 5, certain input/output pins had to be chosen on the Arduino and used as such, as shown in Figure 6. A digital pin had to be set as input for the LTC signal sent from the UltraSync, which in this case was digital pin 8. In order to properly receive the LTC signal from the UltraSync, there were necessary adapters. A DIN cable was first able to transmit the LTC signal from the SYNC port of the UltraSync. The settings for the SYNC port needed to be set to “LTC” mode and the level had to be set to “HIGH.” The LTC signal was then adapted to a BNC cable from the DIN cable, and then adapted to a banana plug so that the signal from the BNC could be easily extracted with wires, with the positive clip of the banana plug wired to digital pin 8 and the negative clip of the banana plug wired to ground.

Once the LTC signal was transmitted into the Arduino, it needed to be decoded. Luckily, I found on an online Arduino forum that someone had posted the code the decode LTC signals in Arduino. I adjusted the sampling rate appropriately, and was able to sample and print the LTC values twice every 5 frames, as shown in Figures 9 and 11. The final Arduino code for this project can be found here: https://github.com/neuromotion/datasync.

Additionally, I created a mock neural data signal in my code and sent this signal into an analog input pin of the Arduino. The Arduino Uno is unable to generate analog signals but it can generate digital signals, so I used one digital pin (digital pin 7 in this case) to generate a fake neural data signal. I made this signal alternate from HIGH to LOW repeatedly at a steady rate using a counter, and the analog reading for this signal was recorded using analog input 2. I also set one of the digital pins (digital pin 6 in this case) to set the LED on or off. For the sake of simplicity (and also because for this proof-of-concept the actual mock neural data readings do not necessarily matter), I set the LED on whenever the mock neural signal generated from the Arduino was set to be HIGH, and I set the LED off whenever the mock neural signal generated from the Arduino was set to be LOW. Thus, the outputs for digital pins 6 and 7 ended up matching in this case.

Results

Figures 8, 9, 10, and 11 show evidence for successful synchronization between the Arduino mock neural data rig and the GoPro cameras. Basically, the universal timecode from the :pulse device jammed into the the Arduino and GoPro using the SyncBac and UltraSync, respectively, corresponds to the same states of the LED light as recorded using the GoPro and set using the Arduino. Thus, both the GoPro and the Arduino appear to have a universal time by which their data can be compared.

Another key takeaway from this project was how the SyncBac devices could now successfully jam their timecodes into the GoPros, thus allowing them to be synchronized along with the other devices tied to the Timecode Systems synchronization system. This had been an issue in the Brown Neuromotion Laboratory for around a year, as it seemed like the GoPro and SyncBac could never properly connect to each other. The problem was resolved by updating the firmware for both the SyncBac and the GoPro. The SyncBac Pro had its firmware updated to v2.07, and the GoPro HERO6 had its firmware updated to Version 2.10. I was given the option to update firmware when I plugged each device into my computer, a 2019 16-inch MacBook Pro, which was how I updated them. The status screen of the SyncBac when it is connected to the GoPro properly is shown in Figure 12.

Analysis

This proof-of-concept showed that data recorded by an Arduino can be synchronized to video data from GoPros. However, it did not necessarily demonstrate that real neural data, which would be collected using, for example, and Open Ephys board or Blackrock Cerebus device, could be synchronized with the GoPro video. For this synchronization to occur, the LTC signal from the UltraSync would need to be collected by the Open Ephys board or Blackrock Cerebus, like they were for the Arduino Uno. Due in part to the COVID-19 pandemic, I did not have any exposure to the Open Ephys board, but Nicole told me that it can have the LTC signal from the UltraSync be jammed into it.

Conclusions

This proof-of-concept shows that the SyncBac Pro and UltraSync ONE devices can be used to synchronize the data acquisition from GoPros and other devices into which an UltraSync can jam its LTC signal. It is demonstrated that the Timecode Systems synchronization devices can be used to synchronize data in electrophysiological recording for research purposes.

Further directions with this project could be to synchronize more Arduino data collection rigs (such as a mock behavioral data rig) as well as to synchronize other types of data, such as audio data. It could be worth investigating whether a microphone could have its signal be inputted into the Arduino’s analog inputs such that actual audio data could be synchronized to video and other behavioral acquisition data.

Something else I could have done would have been to test the drift that occurs between different devices’ clocks, and to compare the drift in timestamps to the timecodes that would be jammed into the devices by the SyncBac and UltraSync. It could be worth testing if it is even necessary to synchronize the data acquisition of the various devices, or if their internal clocks are reliable enough for a certain period of time. It could also be a useful investigation to figure out how long of a period of time the internal clocks would be reliable enough for, such that they do not drift too far away from each other for us to match certain types of data up with each other time-wise.

Acknowledgements

I would like to thank Marc Powell, Nicole Provenza, and Radu Darie for answering all my questions about how their data synchronization systems worked for the Brown Neuromotion Laboratory, and pointing me in the right directions. I especially appreciate how they would respond to my messages even during the COVID-19 pandemic, since this has been a more stressful time and communication is more difficult because we can never be in-person with each other. I would like to thank Prof. David Borton for all his guidance on this project and the opportunity he gave me. I also want to thank Maria Isabel Diaz for all the scheduling she worked out for me, as well as for shipping over the equipment necessary to run this project to my house during the COVID-19 pandemic. The code that I used to help me read in LTC signals from the UltraSync came from here: https://forum.arduino.cc/index.php/topic,8237.0.html. I modified it slightly, but it was very useful.

Figures

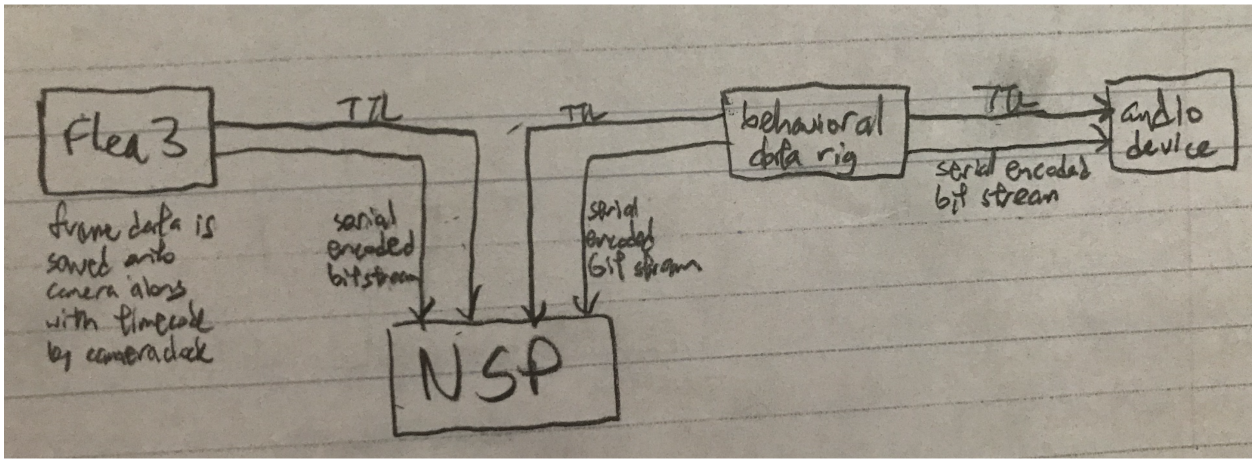

Flea 3
TTL
TTL
behavioral data rig
TTL
audio device
serial encoded bit stream
frame data is saved onto camera along with timecode by cameradeck
serial encoded bitstream
serial encoded bit stream
NSP

Figure 1. Marc's synchronization setup. Serial encoded bit streams representing timecodes are sent into the NSP, which samples data at the highest rate (30,000 samples per second in this case). Thus, there is no reduction in time resolution. Additionally, TTL signals which indicated data capture are sent along to the NSP as well. We can find when data happened relative to the NSP clock based off when the TTL

signal is received by the NSP, finding which serial encoded bitstream (timecode) corresponds to this TTL, and then finding the corresponding data to the timecode.

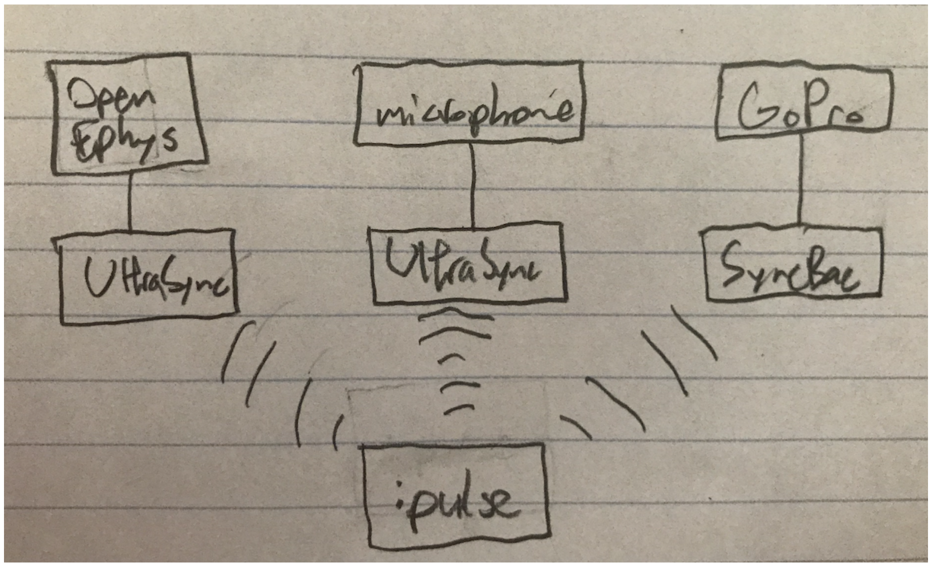


Figure 2. A basic schematic of Nicole's synchronization setup. The :pulse device acts as a master, and sends a universal LTC signal to the UltraSyncs and SyncBacs, which can jam these LTC signals into the Open Ephys, microphone, and GoPros. Thus, we can find when data from all these devices were acquired relative to this universal timecode.

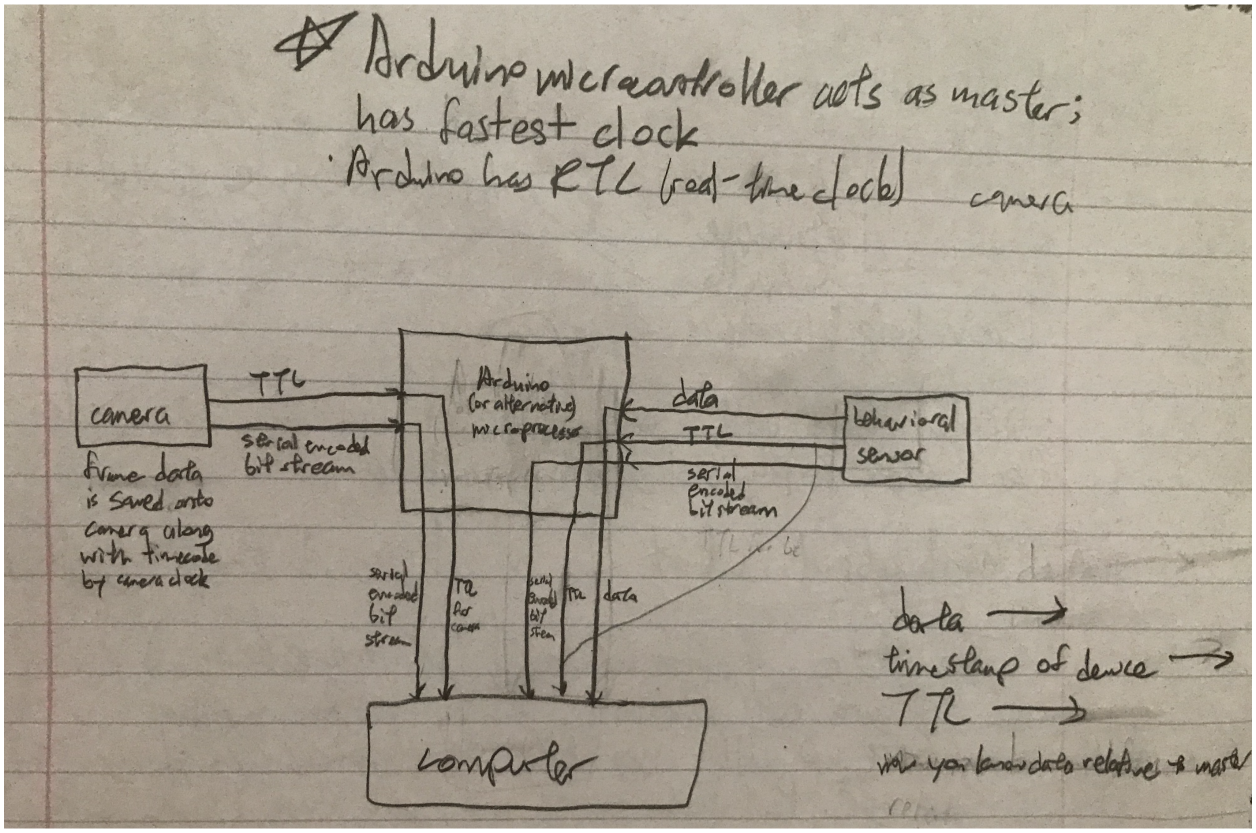
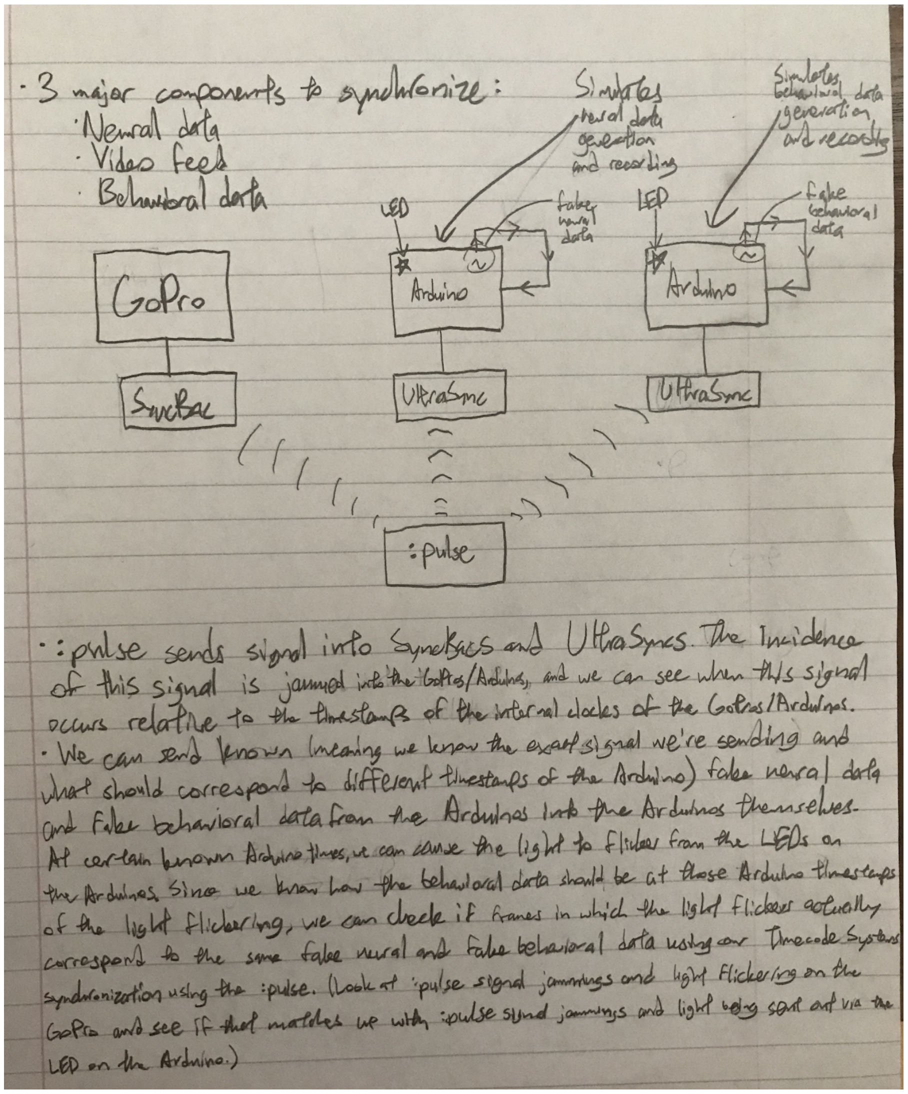

Arduino microcontroller acts as master;
has fastest clock
· Arduino has RTC (real-time clock)
camera
camera
TTL
serial encoded bit stream
frame data is saved onto camera along with timecode by camera clock
Arduino (or alternative) microprocessor
data
TTL
serial encoded bit stream
behavioral sensor
serial encoded bit stream
TTL for camera
serial encoded bit stream
TTL
data
computer
data →
timestamp of device →
TTL →

Figure 3. The schematic I came up with in my first iteration of the project. The goal here was to come up with a universal way of synchronizing the various data collected in the lab. However, realistically, as advised by Prof. Borton, the Arduino would not be able to collect all the signals corresponding to when neural signals happen, because it has too few input channels. Thus, the plan became to design a proof-of-concept for using Timecode Systems' synchronization devices for synchronizing electrophysiological data.



· :pulse sends signal into SyncBacs and UltraSyncs. The Incidence
of this signal is jammed into the GoPros/Arduinos, and we can see when this signal
occurs relative to the timestamps of the internal clocks of the GoPros/Arduinos.
· We can send known (meaning we know the exact signal we're sending and
what should correspond to different timestamps of the Arduino) fake neural data
and fake behavioral data from the Arduinos into the Arduinos themselves.
At certain known Arduino times, we can cause the light to flicker from the LEDs on
the Arduinos. Since we know how the behavioral data should be at those Arduino timestamps
of the light flickering, we can check if frames in which the light flickers actually
correspond to the same fake neural and fake behavioral data using our Timecode Systems
synchronization using the :pulse. (Look at :pulse signal jammings and light flickering on the
GoPro and see if that matches up with :pulse signal jammings and light being sent out via the
LED on the Arduino.)

Figure 4. The proof-of-concept setup I originally came up with for using Timecode Systems' equipment to synchronize various electrophysiological study data acquired in the Brown Neuromotion Laboratory. The idea is that the :pulse device sends a universal LTC signal into the Timecode Systems SyncBac and UltraSync devices. The corresponding LTC signals can be jammed into the GoPro and Arduinos. At the same time, an LED on the Arduinos can flicker at known times in terms of the Arduino clock. If the LED flickers at the same LTC values as those received by the Arduino from the UltraSync and the LTC values of the GoPro received from the SyncBac when it records the LED flickering, then this can indicate that synchronization of acquisition of data occurred.

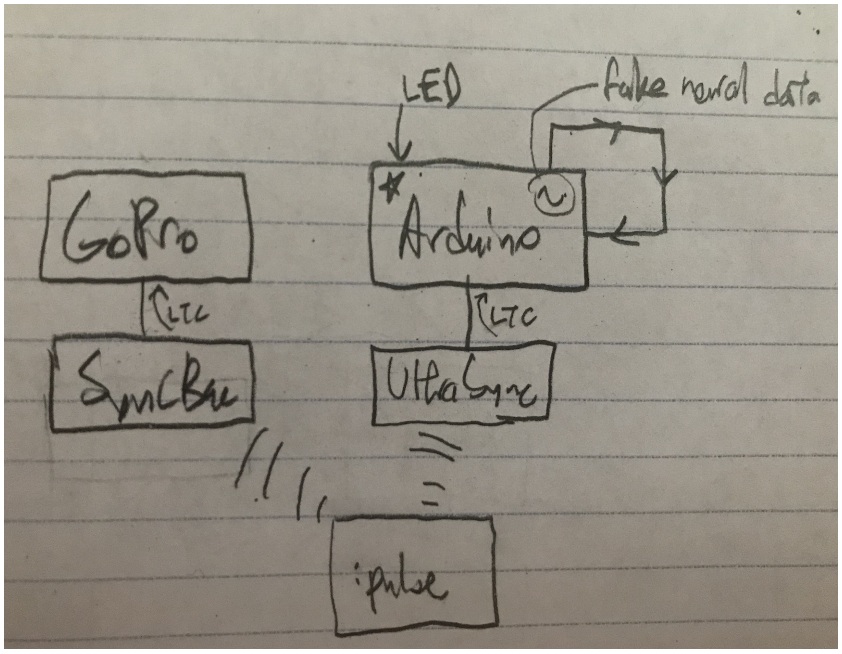


Figure 5. The reduced proof-of-concept setup I came up with, modifying that shown in Figure 4. I thought the setup for the behavioral data mock data capture and neural data mock data capture were redundant (since they were comprised of the same setup for a rig), so I just implemented the use of one Arduino which I chose to represent mock neural data capture to simplify things.

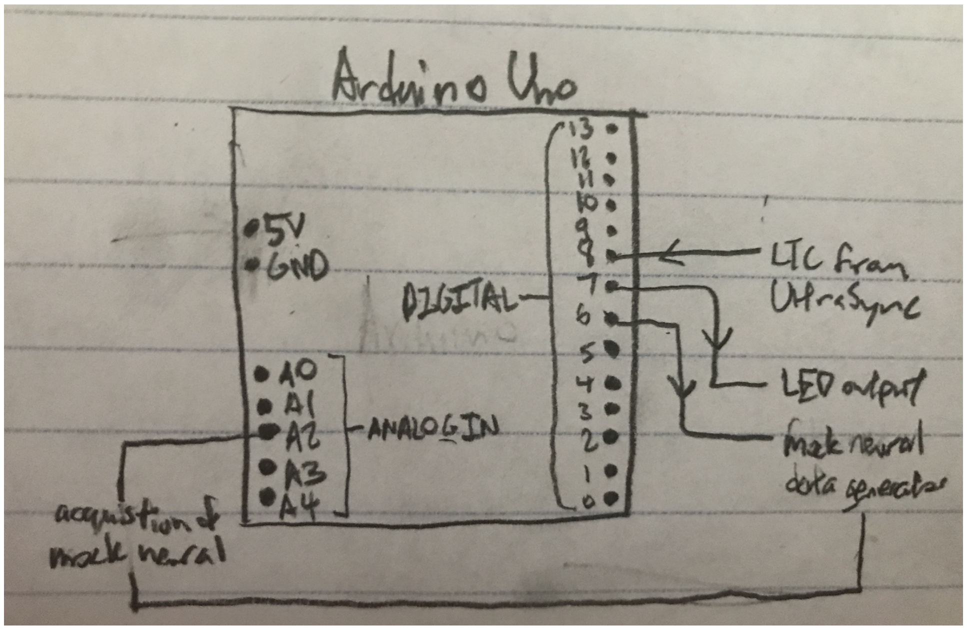


Figure 6. A diagram of the input and output signals of the Arduino Uno. Digital pin 6 was used as output to generate mock neural data, which was recorded as input by analog input pin 2. Digital pin 7 was used to control the flickering of the LED. Digital pin 8 received the LTC signal from the UltraSync into the Arduino.

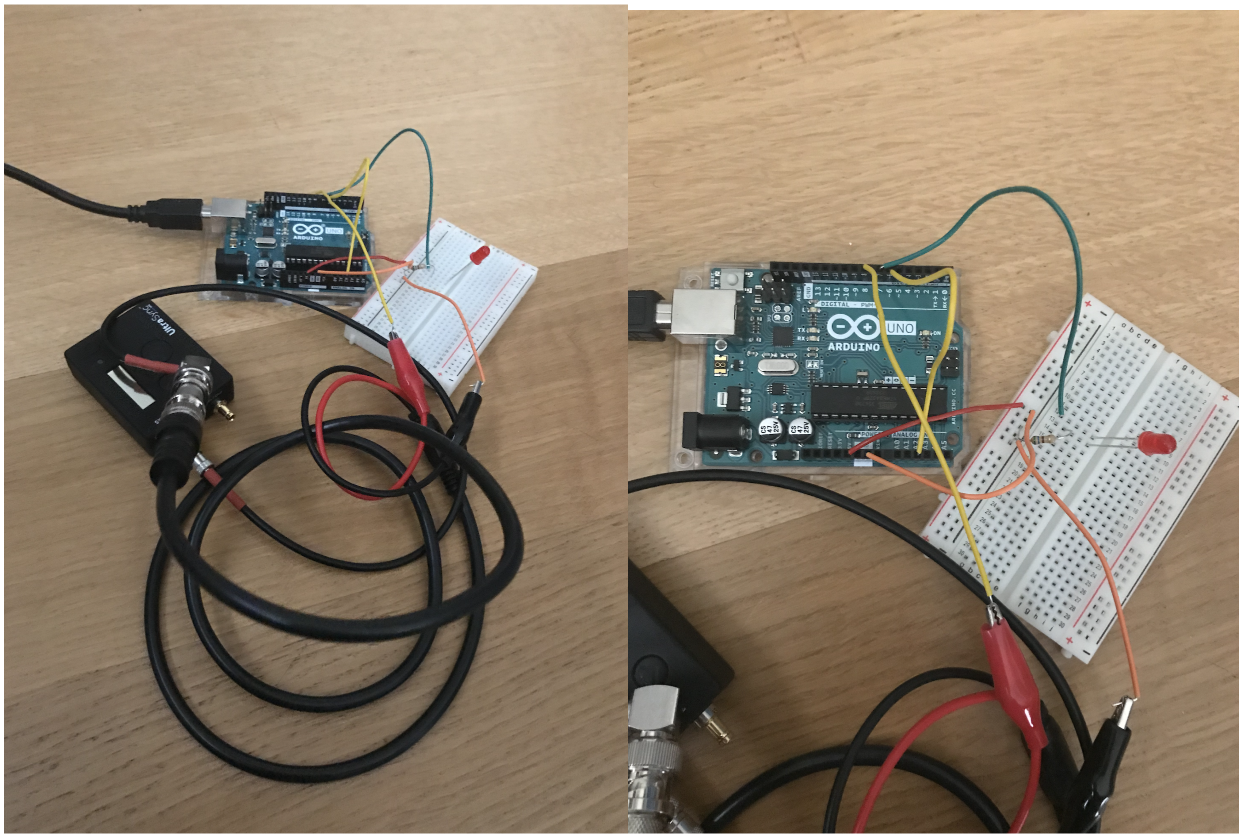


Figure 7. Pictures of the setup I had for the Arduino Uno. A DIN cable was plugged into the SYNC port of the UltraSync in order to transmit its LTC signal. This signal was then adapted into a BNC cable, which then made use of a BNC to banana plug adapter to extract the LTC signal and input it into digital pin 8 of the Arduino.

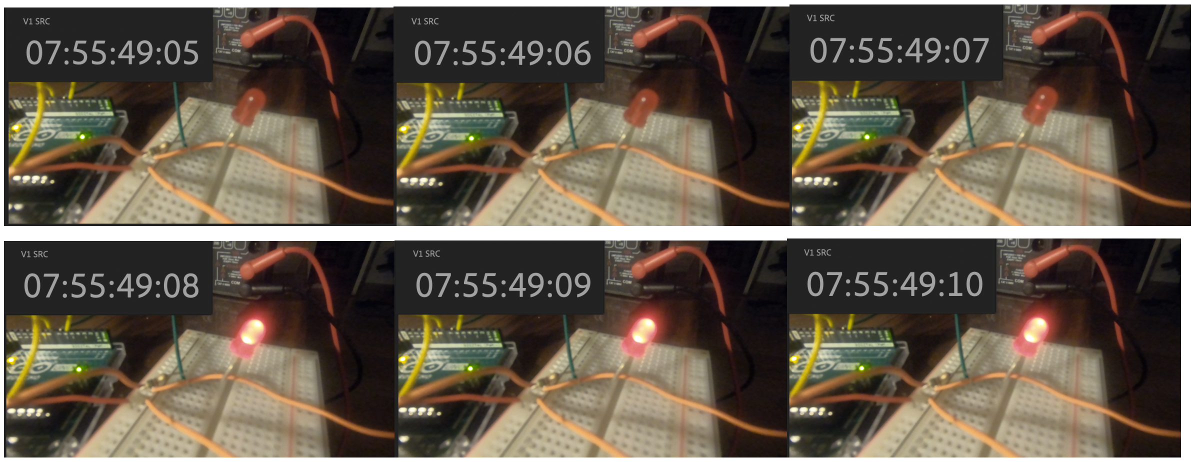
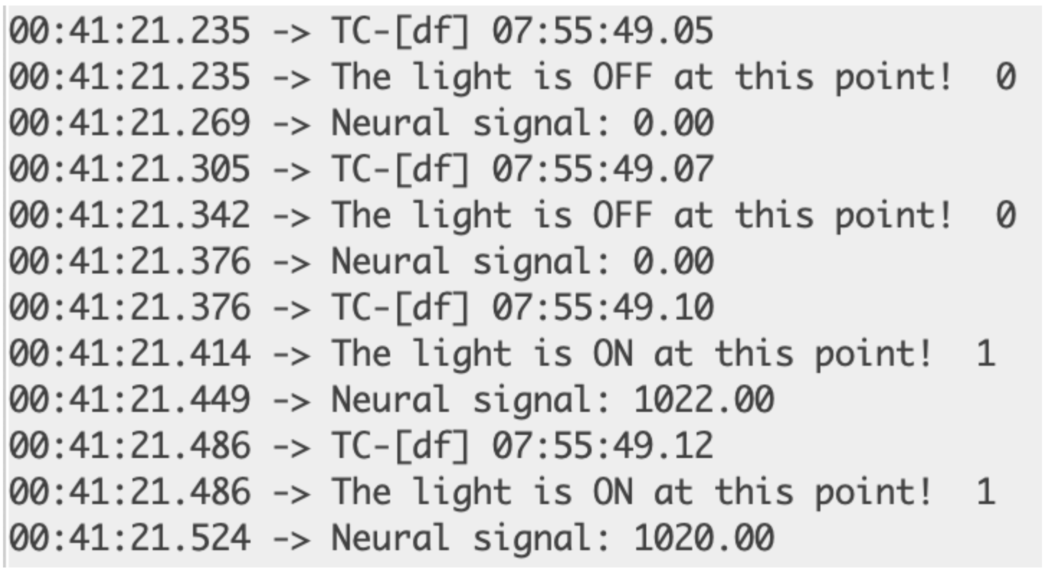


Figure 8. Frame by frame images taken from the GoPro of the Arduino when its LED turns on. The corresponding LTC timecodes jammed by the SyncBac into the GoPro are shown in the upper left of each frame. We can observe that these timecodes

correspond to those listed in the data stream of Figure 9, and that the GoPro frames correctly correspond to when the light of the Arduino is on or off as indicated in Figure 9, signifying that the GoPro and Arduino are properly synchronized in terms of their acquisition of data.



```
00:41:21.235 -> TC-[df] 07:55:49.05
00:41:21.235 -> The light is OFF at this point!  0
00:41:21.269 -> Neural signal: 0.00
00:41:21.305 -> TC-[df] 07:55:49.07
00:41:21.342 -> The light is OFF at this point!  0
00:41:21.376 -> Neural signal: 0.00
00:41:21.376 -> TC-[df] 07:55:49.10
00:41:21.414 -> The light is ON at this point!  1
00:41:21.449 -> Neural signal: 1022.00
00:41:21.486 -> TC-[df] 07:55:49.12
00:41:21.486 -> The light is ON at this point!  1
00:41:21.524 -> Neural signal: 1020.00
```

Figure 9. The data stream printed by the Arduino when the LED turns on. The timecode listed after “TC-[df]” is the timecode that is received from the UltraSync and decoded by my program run on the Arduino. The analog mock neural signals are also listed. These values correspond to the actual voltage (in volts) received at the analog pins multiplied by a factor of 204.8. As we can see, the light is indicated as being on in the Arduino at the same UltraSync LTC’s as for those frames of the GoPro where the light is visibly on (as shown in Figure 8). This shows evidence for successful synchronization.

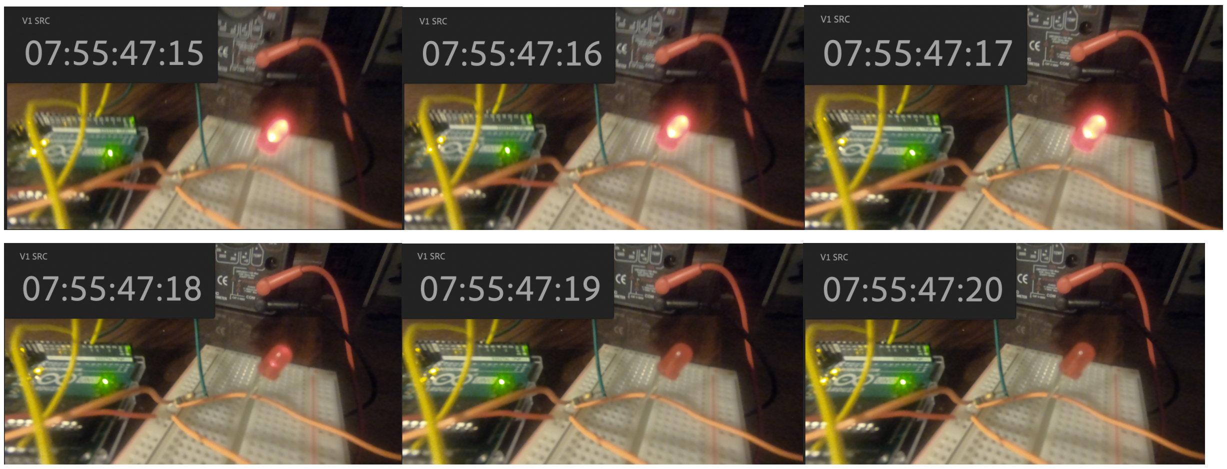
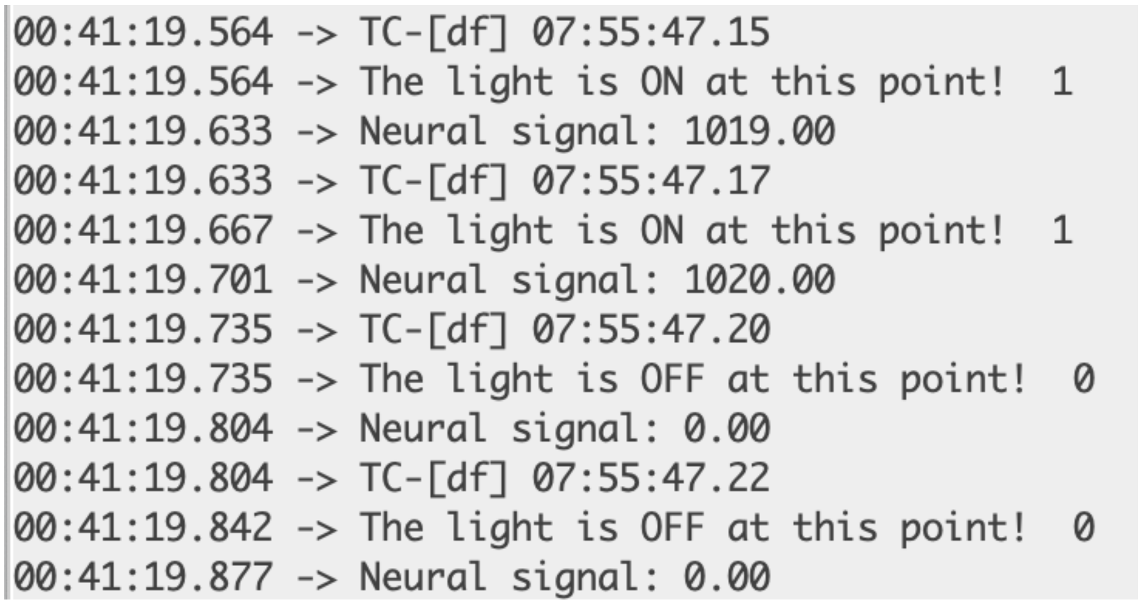


Figure 10. Frame by frame images taken from the GoPro of the Arduino when its LED turns off. The corresponding LTC timecodes jammed by the SyncBac into the GoPro are shown in the upper left of each frame. We can observe that these timecodes

correspond to those listed in the data stream of Figure 11, and that the GoPro frames correctly correspond to when the light of the Arduino is on or off as indicated in Figure 11, signifying that the GoPro and Arduino are properly synchronized in terms of their acquisition of data.



```
00:41:19.564 -> TC-[df] 07:55:47.15
00:41:19.564 -> The light is ON at this point!  1
00:41:19.633 -> Neural signal: 1019.00
00:41:19.633 -> TC-[df] 07:55:47.17
00:41:19.667 -> The light is ON at this point!  1
00:41:19.701 -> Neural signal: 1020.00
00:41:19.735 -> TC-[df] 07:55:47.20
00:41:19.735 -> The light is OFF at this point!  0
00:41:19.804 -> Neural signal: 0.00
00:41:19.804 -> TC-[df] 07:55:47.22
00:41:19.842 -> The light is OFF at this point!  0
00:41:19.877 -> Neural signal: 0.00
```

Figure 11. The data stream printed by the Arduino when the LED turns off. The timecode listed after “TC-[df]” is the timecode that is received from the UltraSync and decoded by my program run on the Arduino. The analog mock neural signals are also listed. These values correspond to the actual voltage (in volts) received at the analog pins multiplied by a factor of 204.8. As we can see, the light is indicated as being on in the Arduino at the same UltraSync LTC’s as for those frames of the GoPro where the light is visibly on (as shown in Figure 10). This shows evidence for successful synchronization.

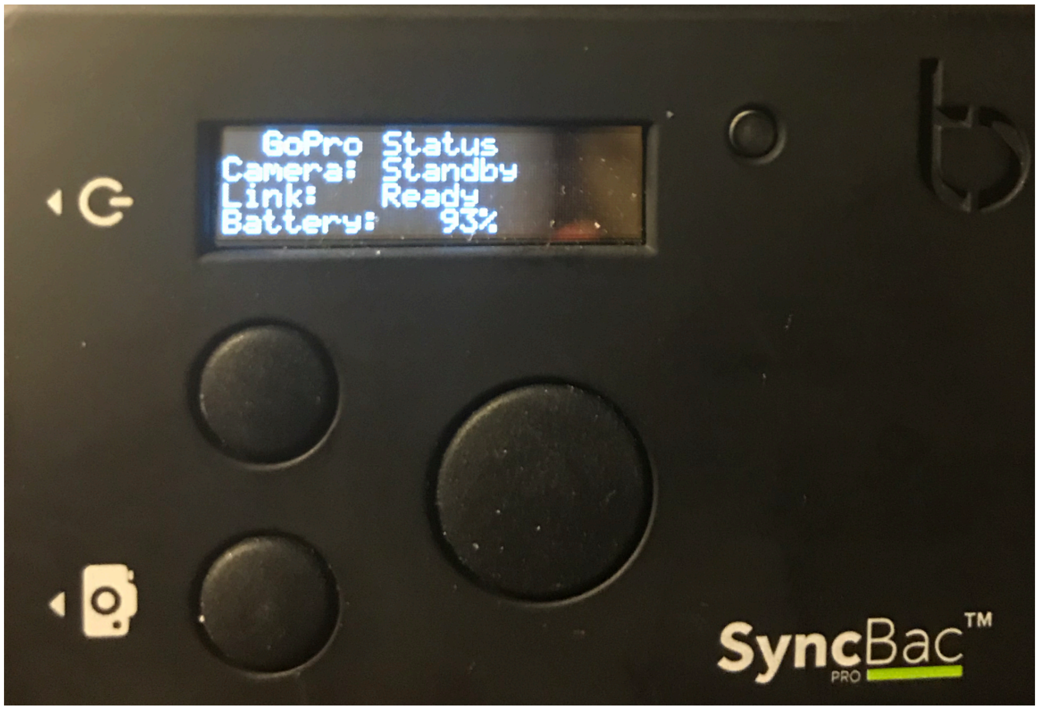


Figure 12. The status screen of the SyncBac Pro when it is properly connected to GoPro HERO6 and can properly jam its timecodes into the GoPro. This happened for me after updating the firmware of both the SyncBac and the GoPro, and thus the GoPro could synchronize with other devices.

References

https://forum.arduino.cc/index.php/topic,8237.0.html
https://www.timecodesystems.com/wp-content/uploads/2017/07/ultrasync-one-user-guide.pdf
https://www.timecodesystems.com/sync-simplified/
https://www.timecodesystems.com/wp-content/uploads/2016/11/SyncBac-User-Guide.pdf
https://forum.arduino.cc/index.php?topic=126677.0
https://support.timecodesystems.com/hc/en-us/articles/115000773992-Ports-UltraSync-ONE
https://www.blackrockmicro.com/neuroscience-research-products/neural-data-acquisition-systems/cerebus-daq-system/
https://flir.app.boxcn.net/s/sm9hdchnm4npy0aoqhmp0ef60yjewey4