Final Report

EpiCap - A Wearable EEG

December 8, 2021

ECE 445 - Fall 2021

Team 9

Shiru Shong (shirus2)

Qihang Zhao (qihangz2)

Casey Bryniarski (bryniar2)

TA: Melia Josephine

With help from Jennifer Cortes and Kenny Leung

Abstract

Our project, EpiCap, was motivated by designing a discreet, portable medical device that monitors a patient's EEG signals from the electrodes placed against his/her scalp. EpiCap continuously processes the EEG data to determine whether a seizure activity is occurring. When a seizure is detected, the device stores this EEG data and records a video of the patient's eye movement from an onboard camera. As both the EEG data and video recordings are stored on an onboard SD card, a physician would use this valuable information to determine a diagnosis. A physician will also be able to visualize the EEG data collected from the EpiCap through open-source software, OpenBCI GUI.

Table of Contents

1. Introduction

1.1 Problem Statement

Electroencephalograms (EEGs) are procedures that measure electrical activity at the very perimeter of the brain. Physicians use the results of these tests to diagnose and determine courses of treatment for abnormal brain behavior, such as epilepsy. A typical EEG test, however, presents difficulties to the patient and physician. Patients can be admitted to a hospital, occupying an inpatient bed, while sleep-deprived and off medication, in hopes that a seizure occurs and can be assessed; which leads to an increase in healthcare resource utilization [1]. As the average hospital stay for epilepsy patients is 3.6 days, the accumulated hospital costs for epilepsy annually totaled approximately $2.5 billion [2]. Additionally, a survey showed that the average annual cost of epilepsy per person was $15,414 [3]; which includes outpatient, inpatient, ED, and treatment costs.

Ambulatory options, which are more cost-effective than outpatient/inpatient treatments, do exist. However, as many of these ambulatory types of equipment are bulky, not portable, or discreet; patients are forced to stay home and surrender important responsibilities that impact the patients' normal everyday activity such as work. People with a history of epilepsy were observed to have a lower annual income and a higher probability of unemployment [4].

1.2 Solution

If the present ambulatory technology was further miniaturized, we can create a device as discreet as a baseball cap that allows the wearers to carry on about their day while remaining monitored for EEG activity. Such a device would eliminate the bulky, inconvenient present ambulatory systems, and draw less attention in public. EEG patients would have the ability to wear it to work and would be able to record important EEG data if a seizure was to occur.

We propose a discrete ambulatory EEG that can monitor patients, come off standby when an event is occurring, and measure brain activity while also utilizing a camera to further record muscle activity (or inactivity), essential data to arrive at a diagnosis and severity of an event. The benefit of having a seizure captured on video is the seamless synchronization to the patient's brain waves recorded from the EEG data. Moreover, the device will include onboard storage that will save EEG data and camera footage from a seizure event. We can then use the onboard SD cards to have the EEG data and camera footage can be viewed on a physician's PC. Our device has three high-level requirements, which were met individually, ensuring that most parts of the device are working:

1. The EEG cap must be discreet and all the main devices components must be within the cap and cap visor (enclosure volume = 72 mm x 36 mm x 25 mm).
2. Record EEG data at 240 +/- 5% Hz sampling rate for at least 24 hours and be able to store EEG data– electrical activity of the brain during a seizure on the flash storage.
3. The EEG cap will track the patient's eye and arm movement to shoulder height by using the wide-angle camera (minimum 240p) located in the cap visor.

The EpiCap requires the following subsystems in order to operate successfully: power subsystem, microcontroller/processing subsystem, onboard flash storage, and a camera module as shown in Figure 1. The power supply would provide the proper 3.3V and 5V to the board to ensure that the system can be running for at least 24 hours. The microcontroller subsystem, which consists of the STM32F2, would be the central processing unit of the system and would be dealing with commands such as saving EEG traces and saving camera footage to onboard storage when detected seizures. The onboard flash storage would store EEG data and eye/arm movement camera footage. Lastly, the camera subsystem consists of a wide-angle camera in order to track the patient's eye and arm movements.

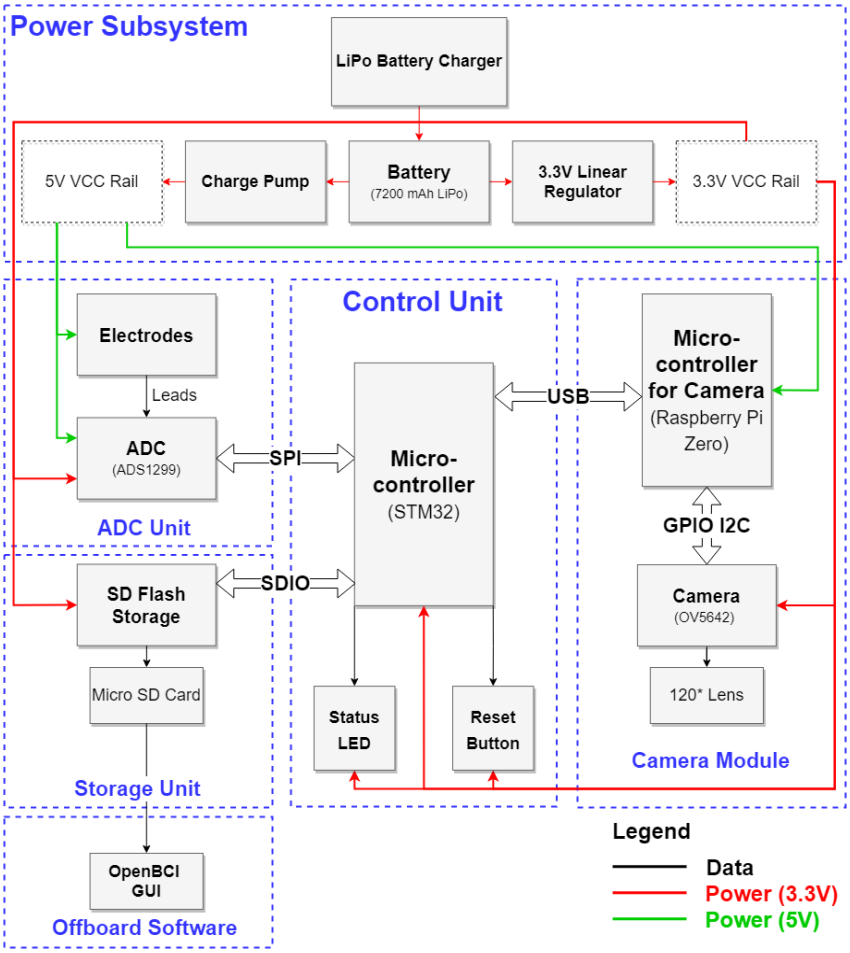

Figure 1. EpiCap Initial High-Level Block Diagram with STM32

As we faced several firmware problems with the STM32F2, we decided to create another board that satisfies our three high-level requirements with Raspberry Pi Zero W as the main microcontroller shown in Figure 2. The main changes include the Raspberry Pi Zero W being directly connected to the ADC unit through SPI and the Camera Module through CSI. We do not require an onboard storage unit as the Raspberry Pi Zero W has its own onboard SD storage. Therefore, all the EEG data and camera footage will be saved to the Raspberry Pi Zero W SD storage.

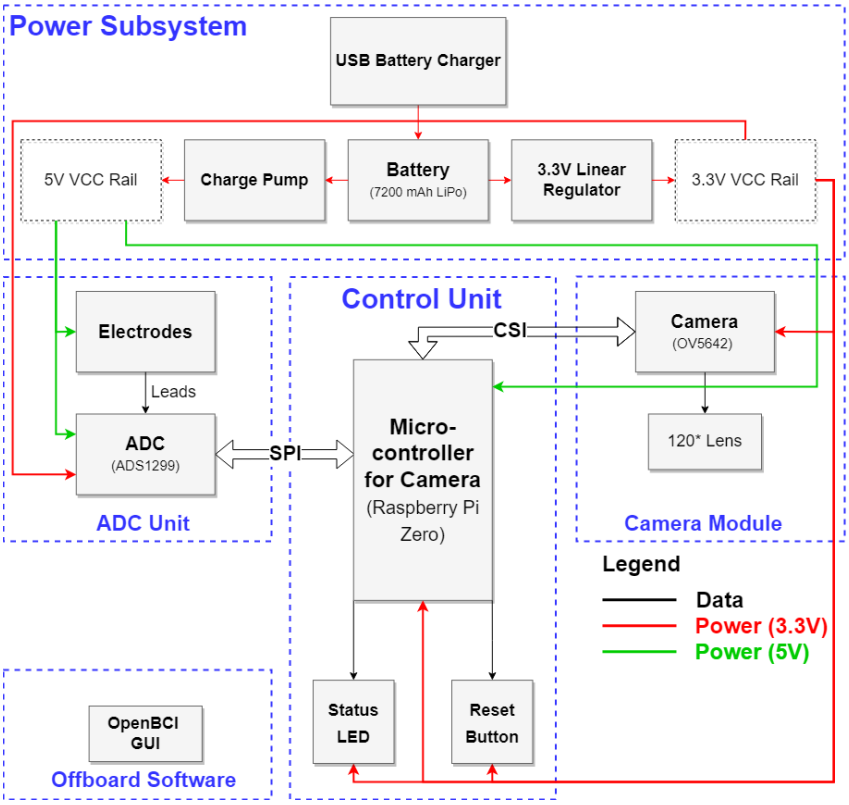

Figure 2. EpiCap Initial High-Level Block Diagram with Raspberry Pi Zero W

2. Design

2.1 Power Subsystem

Our power subsystem prioritizes portability and low noise. Portability is a must since we have to present a device that could remain secure and unobtrusive clipped to the visor of a baseball cap. The economy of scale limited our battery choices to either LiPo, NiMH, or alkaline disposable batteries. Due to their ubiquity, high performance, and high power/weight ratio, we chose a LiPo cell to power our project. Moreover, we sought to find an off-the-shelf solution to reduce our onboard complexity. For this, we chose a USB battery pack, commonly used to charge cell phones in an emergency.

Due to the nature of EEG measurements, we also require a power source that provides a very low noise reference voltage. We compare the signals at the electrodes placed on the scalp to this reference voltage to achieve sub-uV measurements. This eliminates any possible noise present if our wearer's head was a floating voltage. While our reference voltage in this case was 5V, it could have potentially been different had we gone with a different ADC.

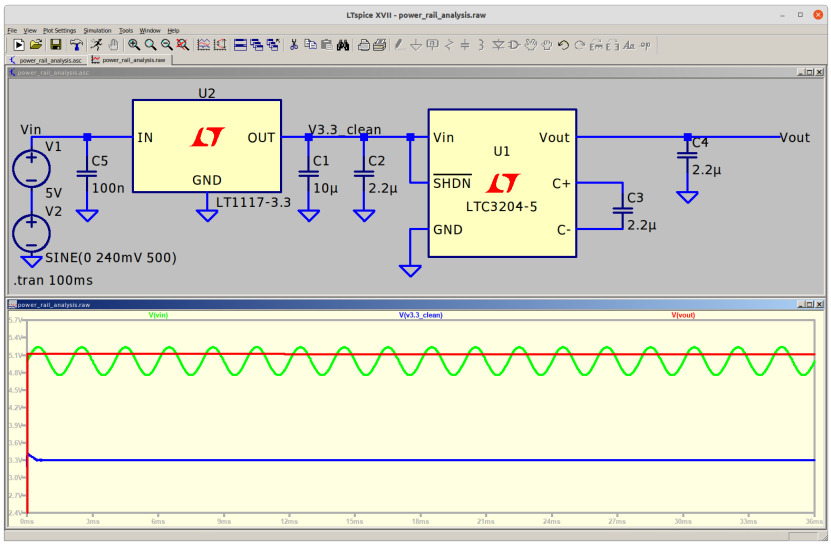

Figure 3. Simulation of our power rail

As could be gathered from the figure, our simulation promises that the linear regulator cancels quite a large bit of noise. We were able to take real-world measurements once our circuit was soldered onto a bare board for unit testing and confirmed that we reached our requirements. While the linear regulator cancelled much of the rechargeable battery pack's noise, we still were only left with a 3.3V rail. In order to create our 5V bias rail, we selected an inductorless charge pump, the LTC3204. We ran our low-noise 3.3V supply from earlier into the input of the charge pump, and were able to confirm that our output in simulation was very low noise, and a major improvement from the original 5V rail we used.

We confirmed functionality some more when we unit tested our power rail on a bare board. Our output from the charge pump was a very low noise, but clean, 5V signal.

2.2 Logic Subsystem

2.2.1 Microcontroller

Our logic subsystem had a few goals. It first had to have enough onboard RAM to handle monitoring 8 channels of 24-bit voltage data at 240Hz. Come time to record, it had to be able to buffer both these data with 240p video data and store all of it on a microSD card.

While very many microcontrollers would and could be used to buffer our ADC voltage data, we found very few that could process our 240p video signal. The 32-bit architecture of the STM32 is suited for arithmetic tasks on our 24-bit EEG data compared to architectures using 8- and 16-bit radixes. Moreover, the STM32 lines offered a proprietary DCMI digital camera interface that had a robust driver and reference implementations that could potentially save us time interfacing our camera submodule.

This potential logic/control unit also required a means of getting data onto an SD card. Many microcontrollers use a scheme known as DMA, or direct memory access, to pull data straight from registers to an output. STM32 implemented the standard SDIO protocol in drivers, and allowed us to use DMA to send data to the SD card directly from our buffers.

Other microcontroller options we looked at were ATMega series ones similar to the one Arduino uses, as well as PIC32MX series microcontrollers. We explored, too, modules that involved the ESP32 line of chips from Espressif. None of these options were able to be as performant as the STM32 option, and all of them required additional memory ICs, DRAM, in order to provide us with enough space to buffer video data. These additional complications led us to pick the STM32 series.

The algorithm behind the STM32 microcontroller is straightforward. We grab 10 sets of EEG data from the 8 channels of the ADC, calculate their average and compare this average with the average of previous 10 sets of EEG data. To calculate the average, we use the equation:

$$Average = \frac{Sum}{Total\ Number} = \frac{\sum|x|}{n}$$

To Compare this average with the previous average, we use the following logic:

- If $Current\ Average \simeq Previous\ Average$:
  The patient is not in a state of seizure.
- If $Current\ Average$ > Threshold*($Previous\ Average$):
  The patient is in a state of seizure.

2.2.2 Python API and Raspberry Pi

We also have our backup PCB board which uses the Raspberry Pi as a microcontroller to communicate among ADS1299, itself, and camera through the SPI port. The Python API deals with the SPI port and uses another GPIO thread to process the samples from ADS1299. Those samples are managed into a NumPy array by the registered callback and are processed to the length corresponding to the number of channels. Even though they provide these arrays to us, we still need a FIFO buffer to store these data, since we need to store 2 arrays into the SD filesystem to do the comparison. We also import the os library to make a direct system call to call the camera module, output the footage, and save it in the date-time format in a folder called Videos into the SD card.

2.3 Storage Subsystem

Our storage subsystem had to prioritize usability and familiarity - this was one of the few parts of our device that would be regularly handled by patients and physicians. Due to their ubiquity and small size, microSD storage cards were far and away our first choice for onboard storage media. They provided a safer means of privacy than a wireless alternative, offered very high capacity at low prices (less than $1 per GB of storage), and due to their use in phones, computers, and tablets, end users wouldn't require much additional training to properly extract test data from our device. Other alternatives present were USB mass storage devices, and larger SD cards, but they posed a threat to our device's small footprint and any performance increase they could offer was out of the scope of our device.

Our STM32 firmware used a reference implementation, instantiating a FAT16 filesystem on our SD card before writing data to it. FAT16 is a common, small footprint filesystem that offers portability and cross-compatibility with all major operating systems, affording our data to

be read by physicians with Mac, PC, or Linux computers. Our 24-bit ADC data would be sent from an on-board buffer and written to a text file, while our 240p video signal would be written frame by frame to a headless video file, like h.264 format.

2.4 Sensor Subsystem

Our sensor subsystem had three tasks to take: it must implement a way to communicate with and be controlled by our logic unit, it must be able to measure and discriminate ~uV EEG signals, and it must allow us to sample at a rate at least at 240Hz. Due to the unique use case of our project, we required a "biopotential ADC" that offered a reference voltage or biasing output as well, in order to actually perform an EEG reading.

EEGs take readings from electrodes placed on the patient's scalp. If the patient's head was a floating voltage, the electrodes wouldn't have any consistent signal to discriminate against. Instead, EEGs always have a few electrodes, in our case, ear clips, that supply this reference voltage that is later subtracted to achieve our EEG measurements. It is integral that we have this bias, or else we face taking inaccurate measurements from the wearer [6].

We explored our options for biopotential ADCs in Table 1 below. TI's ADS1299 was the only IC that didn't require additional amplifiers, isolation ICs, or additional packages in the reference layout of the device. We determined that the ADS1299 would be performant enough for our needs, since it is also used by the OpenBCI project to achieve research-grade EEGs, while providing us with the most efficient use of space on the PCB in LQFP package size.

Table 1. EEG-capable ADCs

Manufacturer [6][7][8]	Package Options	Sample resolution
Texas Instruments ADS1299	LQFP	24 bit resolution
Analog AD7177-2	TSSOP	24 or 32 bit resolution
Maxim Integrated MAX11040K	TSSOP	24 or 16 bit resolution

Since all of the chips we looked at implemented a SPI communication/control bus, this requirement ended up being a moot point. We didn't think we would have much of an issue comparing the devices in this aspect, since SPI is an industry-standard and our ADCs operated at bandwidths far below what SPI is capable of.

While we were able to pick an ADC that didn't require much conditioning, we still were encouraged to isolate each lead with bias capacitors per TI's reference implementation. We also drew heavily from OpenBCI's implementation of the ADS1299, which proved to be a successful product that had full functionality while remaining in a small footprint. For this, we chose an RC circuit that provided a low-pass filter in order to prevent spurious discharges from reaching our sensitive sensors.

No matter what option we went with, ADCs are extremely sensitive to electrostatic discharge [9]. Static electricity is generated by a person's clothing, and the stored charge is released when a person touches a ready path to ground, or handles electronics similar to ours. We mitigate this hazard through TVS diodes. While there were plenty of options available, we went with TI again for their small footprint TVS diodes, each rated to burn at 12V. These diodes provided us with enough protection that, in the event of a wearer or someone else having a static charge, could burn the TVS diode without harming the $40 ADC onboard the device. The TVS diode would also pose much less of a stress to repair compared to reworking the 64-pin ADC package.

2.5 Camera Subsystem

The camera module, which consists of an OV5647 Wide Angle FOV 160° 5-Megapixel camera, would be directly connected to the Raspberry Pi Zero W microcontroller through the CSI port. The Python program on the Raspberry Pi Zero W would trigger the camera to turn on and start recording when detected a seizure. The camera module would send the processed camera footage to be stored on the onboard SD card of the Raspberry Pi Zero W (Appendix C, Figure XX). The OV5647 camera module has a maximum transfer rate of 60 fps and VGA resolution (640x480). The camera footage will be stored onto the onboard storage of the Raspberry Pi Zero W. To allow ease of use by the physician, the camera footage should be written to a file that can be easily viewed by an open-source multimedia player.

3. Verification

3.1 Power Subsystem

The power subsystem sets specific ripple voltage tolerances for each power rail (Appendix A, Table XX.1). We populate and solder only the components essential to the power rail and then measure each output with a multimeter. We then use an oscilloscope to measure our 5V input to GND, to determine the baseline ripple. We then measure our 3.3V output from our linear regulator to our GND, and use an oscilloscope to ensure our output voltage and ripple are within our tolerances. Finally, we measure the voltage across the output capacitors on the charge pump, to verify we have a stable 5V rail, and determine whether our ripple voltage on that rail is within tolerances via an oscilloscope.

3.2 Logic Subsystem

3.2.1 Microcontroller

We first verify our STM32 functionality by connecting our SWDIO, GND, SWCLK, 5V, nRST, and SWO pins from our modified ST Link v2.1 to the corresponding headers on the board. We flash a firmware payload to the device and determine if the IC is functioning. We then use the SWO/SWDIO trace feature to verify individual parts of our firmware as they are executed by the chip.

3.2.2 Python API and Raspberry Pi

The requirement for the python API was that we must be able to ssh onto the Pi and get the demo ADC data. By compiling the code and booting the whole system up, we were able to get the EEG data in a form of array, and store these data into a txt file named with year-month-day-hour-minute-second. On the other hand, the camera module successfully receives the “turn on” signal from the Raspberry Pi, and saves the footage into a trg file named with year-month-day-hour-minute-second.

3.3 Storage Subsystem

3.3.1 microSD Card - STM32

The requirement of our storage was to be able to store EEG and video data to an onboard SD card. Our STM32 board firmware was able to write an empty FAT16 filesystem to the SD card, but not achieve communication with the ADC, and unable to write data to the microSD card. We confirm that the FAT16 filesystem is being created by our device by inserting a wiped (blank) microSD onto the carriage present on the STM32 board. We can then run the firmware for a few minutes, making sure our filesystem firmware executes in our software trace. We then check the microSD card on a laptop to determine whether a FAT16 file is present, using lsblk or a similar utility.

3.3.2 microSD Card - Raspberry Pi

Our microSD card onboard the Pi served as both the storage for our host OS, our boot partition, and our space for saved data. We were able to write Python code that saved video data in h.264 format to this onboard SD card. We were also able to save test ADC data from our ADS1299 over SPI.

3.4 Sensor Subsystem

3.4.1 Electrodes

We verified the functionality of our electrodes and headset using the OpenBCI Cyton board. We were able to establish a connection and collect data from the headset and visualize it using the OpenBCI software suite.

3.4.2 ADC

Our ADC could be split into analog and digital components. Our analog side connects to our electrodes and headset, and the digital side communicates with our logic subsystem over SPI. To verify the digital component, we sent a test stream instruction over SPI to our ADC chip. We were able to retrieve a stream of test data to confirm that we could talk to the chip. To verify our analog component, we connected our OpenBCI headset and electrode leads to our protected ADC headers, and compared our measurements with the ones retrieved by the OpenBCI Cyton board. Unfortunately, due to issues with our biasing circuit, we weren't able to get accurate EEG results when the entire project was connected together.

3.5 Camera Subsystem

The requirements for the camera module were of the following: the camera module must have a minimum of 60 fps and 240p resolution (Appendix A, Table XX.1), be able to track both eye movements and arm movements above shoulder height (Appendix A, Table XX.2), camera module size must be less than 40x40 mm and weigh less than 30g (Appendix A, Table XX.3), and the camera footage must be stored onto the onboard SD card with the correct file extension in the respective folder (Appendix A, Table XX.4). To verify that the camera has 60 fps and 240p resolution, we chose the OV5647 camera module which allows us to configure the camera to record 640x480p at 60 fps. The datasheet states that the size of the camera is 24x9 mm and the weight is less than 2g, which meets our specifications. Once we ran the Python program, we were able to verify that the wide-angle camera was able to record the patient's eye movements and arm movements above shoulder height (Appendix C, Figure XX). Once the video recording is taken, we were also able to verify that the output file has the extension of '.h264' by reading the SD card on the laptop and that the file is saved into a Videos folder on the SD card (Appendix C, Figure XX).

4. Cost

4.1 Parts

Table 2. Cost of Each Component Used for EpiCap

Component	Quantity	Cost per Component	Total Cost
Raspberry Pi Zero W	1	$5.000	$5.000
B006603 ArduCam Pi Zero OV5642 Camera Module	2	$17.990	$35.980
STM32F205ZCTx MCU	4	$13.730	$54.920
LD1117S33TR_SOT223 Linear Regulator	5	$0.438	$2.190
LTC3204EDC-5#TRPBF	2	$4.270	$8.540
ADS1299IPAG 8 Channel ADC	1	$70.880	$70.880
DM3BT-DSF-PEJS microSD Card Carriage	2	$2.300	$4.600
SDSDQM-032G-B35 microSDHC card, 32G	2	$5.850	$11.700
Single-cell LiPo battery, 5000 mAh	1	$13.390	$13.390
1568-PRT-15217-ND LiPo Battery Charger	1	$9.100	$9.100
W3A45C102MAT2A 1000pF Capacitor Array	10	0.248	$2.480
RC1206FR-0710KL 10 kOhm Resistor	10	0.074	$0.740
TPD4E1B06DCKR TVS Diode	12	0.56	$6.720
CAY16-2201F4LF 2.2kOhm Resistor Array	10	0.086	$0.860
RC1206FR-07680RL 680 Ohm Resistor	10	0.074	$0.740
RC1206FR-07220RL 220 Ohm Resistor	10	0.074	$0.740
CRT0805-BY-1004ELF 1MOhm Resistor	10	0.179	$1.790
RC0805JR-0747KL 47 kOhm Resistor	10	0.036	$0.360
C0805C225K8RAC7210 2.2 uF Capacitor	10	0.162	$1.620
0805AC101KAT2A 100 pF Capacitor	10	0.54	$5.400
CC0805BRNPO9BN2R2 2.2 pF Capacitor	10	0.124	$1.240
C0805X104K1RACAUTO 0.1 uF Capacitor	50	0.2144	$10.720
C0805C106K8PAC 10 uF Capacitor	10	0.632	$6.320
C0805X105K5RACAUTO 1 uF Capacitor	30	0.86	$25.800
PMK212BBJ107MG-T 100 uF Capacitor	10	1.13	$11.300
C0805X102K5RAC3316 1000 pF Capacitor	10	0.193	$1.930
2223 Raspberry Pi Header Pin	2	2.5	$5.000
SMLMN2ECTT86C SMD LED	5	0.75	$3.750
		Total:	$303.810

4.2 Labor

From the 2019-2020 annual Illini Success Report, an electrical engineer has an average salary of $76,129 while a computer engineer has an average salary of $99,145 [5]. Assuming a total of 100 billable hours per engineer as there are 10 weeks we worked on the technical part of the project and 10 hours per week. We would also require an additional $125/hr for our company's upkeep. As Casey and Shiru are electrical engineer students while Qihang is a computer engineering student, the total labor cost would be the following:

$$Total\ Labor\ Cost\ for\ 2\ Electrical\ Engineers\text{: } \$36.63/hr\ \times 100\ \times 2 = \$7,326$$

$$Total\ Labor\ Cost\ for\ 1\ Computer\ Engineer\text{: } \$47.67/hr\ \times 100\ = \$4,767$$

$$Total\ Labor\ Cost\text{: } 7326 + 4767 + 125(100)\ = \$24,593$$

As we did not require the machine shop's assistance, the total labor cost would be **$24,593** for 3 works in total.

4.3 Schedule

Table 3. Project Schedule and Task Allocation for Each Worker

Week	Shiru Shong	Casey Bryniarski	Qihang Zhao
9/27	Complete Design Document.	Complete Schematic for EpiCap.	Complete Design Document.
10/4	Complete Design Review. Get the PCB board approved and order first around. Buy all the necessary components for the project. Test OpenBCI EpiCap.	Complete Design Review. Finalize & Review PCB. Get the PCB board approved and order the first round. Test OpenBCI EpiCap.	Complete Design Review. Finalize & Review PCB. Buy all the necessary components for the project. Test with OpenBCI GUI.
10/11	Complete unit-testing for each component. Complete PCB Board by soldering and mounting components.	Complete unit-testing for each component. Complete PCB Board by soldering and mounting components.	Work on the firmware of the microcontroller with other subsystems.
10/18	Test onboard hardware system, identify and fix issues. Start Version 2 PCB design.	Test onboard hardware system, identify and fix issues. Start Version 2 PCB design.	Finish the firmware code for the microcontroller.
10/25	Finish on-board hardware testing. Execute the whole EpiCap testing, including firmware. Order Version 2 PCB board design.	Finish on-board hardware testing. Execute the whole EpiCap testing, including firmware. Order Version 2 PCB board design.	Test firmware code with the entire hardware component. Start Version 2 firmware coding.
11/1	Complete the Version 2 PCB board and perform the final debugging process.	Complete the Version 2 PCB board and perform the final debugging process.	Finish Version 2 firmware coding and continue the firmware debugging process.
11/8	Perform final complete system testing. Prepare for Mock Demo and demonstration.	Perform final complete system testing. Prepare for Mock Demo and demonstration.	Perform final complete system testing. Prepare for Mock Demo and demonstration.
11/15	Mock demo.	Mock demo.	Mock demo.
11/22	Fall break.	Fall break.	Fall break.
11/29	Demonstration. Work on presentation. Start the final paper.	Demonstration. Work on presentation. Start the final paper.	Demonstration. Work on presentation. Start the final paper.
12/6	Final presentation and final paper.	Final presentation and final paper.	Final presentation and final paper.

5. Conclusion

5.1 Accomplishments

Although we were not able to achieve all of our high-level requirements, we were able to successfully complete 2 out of 3 of them. Not only that, we were able to demonstrate that each individual module is fully functional through unit testing. One of the biggest accomplishments was overcoming the challenge of getting the first PCB with the STM32F2 hardware system to work. We had to solder a handful of components that were difficult to solder such as the LTC3204 charge pump and the STM32F2 chip. We also had to cut traces on the board while jumping wires as we discovered a number of PCB trace errors. In the end, we were able to successfully flash the firmware to the STM32F2 board, which verifies that our hardware system works. When we flashed the firmware, we were able to observe register values changing in the STM32, which is a good sign of the firmware being partially functional. One of the other accomplishments is that our camera module is fully functional as the Raspberry Pi was able to communicate with the camera and record the video successfully. Due to an error in our biasing circuit, we were not able to receive accurate EEG signals from the electrodes connected to the ADC unit. However, the Raspberry Pi was still able to successfully communicate with the ADC unit and receive test data to be stored on the SD card. We believe that we are one board iteration away from having a fully functional EpiCap device due to our biasing circuit error.

5.2 Uncertainties

Due to the time constraints within the semester, we were unable to test our project against real-world and clinical examples of EEGs, and EEGs during seizure symptoms. Our detection scheme is a placeholder that, through tweaks and iterations against clinical cases, would give way to a more robust and proven detection scheme. We are confident that our initial detection scheme we describe could remain mostly intact, save for a few tweaks to our buffer's length and our threshold we set to distinguish seizure vs. resting EEG activity.

5.3 Ethical Considerations

There could be several concerns regarding our project. The cap confronts several risks and vulnerabilities as a result of the use of rechargeable lithium-ion batteries, electrodes, cables, chips, flash storage, camera, and the possibility of a patient's unexpected fall.

First, there are electrodes that remain in contact with the scalp in the event of a seizure. Even though this is a special hat that collects medical EEG information, we still need to make sure patients do not feel any difference or discomfort wearing the hat compared to a regular hat. As stated in the subsystem requirements, the electrodes must be readily adapted to different ball caps and different hat sizes. We must also take precautions that our device does not create more

danger for a patient in the event of a fall. In order to solve these issues, we will adjust the gap between the cap and the patient's head by using one of the most common ways of fixing an ordinary cap: an elastic strap, which can better keep the cap on the patient's head steadily and capture important EEG information in the scenario that the patient falls. We will also line up our wires and chips so that they are distributed around the edge of the hat so that the patient will not be injured by these parts if they fall.

The second concern is that EpiCap is a wearable gear, and patients will need to wear it for long periods to get complete EEG data. If any materials are mixed with chemicals that are harmful to the human body, it is potentially dangerous for the patient and may lead to an erroneous diagnosis from doctors. In order to solve this problem, we need to ensure that both the battery and board present no hazard to the patient - especially in the event of high heat, moisture, and any sort of mechanical shock where according to the IEEE Code of ethics I.1: we must "hold paramount the safety, health, and welfare of the public…" [10].

The third concern is that the EpiCap will collect highly confidential EEG data of the patient. We must ensure that patient information can only be accessed by the doctor and is kept confidential to visitors, as according to the IEEE Code of ethics I.1 we must "hold paramount the safety, health, and welfare of the public…"[10]. In order to solve this issue, we can design a password system for doctors and encrypt all patient information. For example, each doctor will have separate accounts and passwords. When the doctor receives data from the GSM chip, they will give their patient an identification number. As a result, only the corresponding physician has the patient's EEG data, which cannot be viewed by the outside viewer.

The fourth concern is that the EpiCap will not have the danger of electrocution from the electrodes. We must ensure that no patients or doctors get hurt according to the IEEE Code of ethics I.1 we must "hold paramount the safety, health, and welfare of the public…"[10]. In order to solve this issue, we can attach a layer of insulation material to the inside of the hat, to avoid direct contact between any electrodes to the skin of patients.

We will rigorously test our design to ensure that the final product is safe for patients and doctors alike. We intend to comply with the IEEE code and the corresponding safety or regulator standards such as OSHA or FCC. Additionally, we will seek and accept any improvements regarding our project and due to the nature of our design, we will need to work with the medical school, so that we will appropriately credit others' efforts, according to IEEE code of ethics I.5: "to seek, accept, and offer honest criticism of technical work…" [10].

5.4 Future Work

We believe that if we correct our biasing circuit, we could produce a board that fits neatly on our Raspberry Pi Zero W and performs our EEG measurements. If this is accomplished, we may begin testing the device and the software detection scheme in a clinical setting.

6. References

[1] S.-Y. Chen, N. Wu, L. Boulanger, and P. Sacco, “Antiepileptic drug treatment patterns and economic burden of commercially-insured patients with refractory epilepsy with partial onset seizures in the United States,” *Journal of Medical Economics*, vol. 16, no. 2, pp. 240–248, 2012. [Accessed September 25, 2021].

[2] Healthcare Cost and Utilization Project, 2014 National Data. [Online]. Available: hcupnet.ahrq.gov/. [Accessed September 25, 2021].

[3] J. A. Cramer, Z. J. Wang, E. Chang, A. Powers, R. Copher, D. Cherepanov, and M. S. Broder, “Healthcare utilization and costs in adults with stable and uncontrolled epilepsy,” *Epilepsy Behav.*, vol. 31, pp. 356–362, 2014. [Accessed September 25, 2021].

[4] Kobau R, Zahran H, Thurman DJ, et al. Epilepsy surveillance among adults—19 states, Behavioral Risk Factor Surveillance System, 2005. *MMWR Surveill Summ.* 2008;57(6):1-20. [Accessed September 25, 2021].

[5] Texas Instruments, “ADS1299-x Low-Noise, 4-, 6-, 8-Channel, 24-Bit, Analog-to-Digital Converter for EEG and Biopotential Measurements,” ADS1299 datasheet, Oct 2016 [Revised Jan. 2017].

[6] Maxim Integrated, “24-/16-Bit, 4-Channel, Simultaneous-Sampling, Cascadable, Sigma-Delta ADCs,” MAX11040K datasheet, May 2015.

[7] Analog Devices, “32-Bit, 10 kSPS, Sigma-Delta ADC with 100 µs Settling and True Rail-to-Rail Buffers,” AD7177-2 datasheet, Mar. 2015 [Revised Mar. 2016].

[8] TI Precision Labs, “EOS and ESD on ADC,” Texas Instruments, https://training.ti.com/eos-and-esd-adc. Accessed 6 September 2021.

[9] “Part of University of Illinois,” Box. [Online]. Available: https://uofi.account.box.com/login. [Accessed: 08-Dec-2021].

[10] IEEE, “IEEE code of ethics,”. [Online]. Available: https://www.ieee.org/about/corporate/governance/p7-8.html. [Accessed: 28-Sep-2021].

Appendix A - Requirement and Verification Tables

Table 4. Requirements and Verification Table for ADC

Requirements	**Verification**	**Verification Status**
1. Must be sampling at a rate of 240 +/-5% Hz.	1. a. Send ADS command to stream 240Hz test data over SPI, and confirm 240Hz functionality with timestamps	Verified
2. Must not pick up any external signals from other peripherals on the board.	2. a. Perform an EEG test with leads, but no head, in the hat, for a long duration, in a reasonably EM-free area. b. While recording data, poll every other peripheral in sequence, for a minute or so each, making sure there's another minute of dead time in between devices running. c. View the data. Identify and determine any concerning, spurious, emissions.	Not verified
3. Provide reasonably low noise (<μV).	3. a. Provide a low-power analog signal to the ADC in the range of μV. b. Take oscilloscope measurements to	Not verified

	measure the output voltage ripple signal is less than 1 μV.	

Table 5. Requirements and Verification Table for Electrodes

Requirements	**Verification**	**Verification Status**
1. Must remain in contact with the scalp in the event of a seizure	1. a. Wear the EpiCap with all the electrodes in contact with the scalp. Confirm that all the electrodes do detect electrical signals initially, using a continuity check on a multimeter. b. Move around and re-enact big physical movements such as falling, jumping, walking, etc. c. Confirm that all the electrodes are still in contact with the scalp and collect electrical signals from the brain accurately.	Verified

2. Must be collecting accurate electrical signals (can have +/- 5% error).	2. a. Provide a low-power analog signal, generated from the function generator to the one electrode lead of the original OpenBCI Ultracortex Mark IV Cap. b. Use the OpenBCI GUI software to record the EEG data received by the Cyton 8-channel board. c. Repeat steps A and B with our own EpiCap board. Ensure that the EEG data collected from our board stays within the 5% error of the data collected from the OpenBCI Cap. d. Repeat steps A, B, and C with the 7 other electrodes used for the 8 channel electrode input.	Steps a. and b. verified

Table 6. Requirements and Verification Table for Rechargeable Lithium Polymer Battery

Requirements	**Verification**	**Verification Status**
3. During discharging at maximum current and voltage, the temperature of the Li-Po battery would be less than 27℃.	3. a. Charge the Li-Po battery entirely to 4.2V to ensure that the cell voltage is at its highest. b. Discharge the battery and use an IR thermometer to ensure that the battery does not discharge at a temperature greater than 27℃.	Verified
4. Must be able to power the board at full capacity for at least 24 hours.	4. a. Connect the fully charged LiPo battery to the board with all the devices connected. b. While connected to the board, discharge the LiPo battery at an operating voltage of 3.5-4.2V for 24 hours. c. Use a voltmeter to ensure that the battery cell voltage drop is greater than 3.2V (preferably between 3.5-3.7V).	Verified

Table 7. Requirements and Verification Table for Linear Regulator

Requirements	**Verification**	**Verification Status**
1. Provide 3.3V+/- 5% from a 3.5-4.2V source.	1. Measure output voltage using an oscilloscope to ensure that the output voltage stays within 5% of 3.3V	Verified
3. Maintain a temperature range of 20℃ to 27℃.	3. Use the IR thermometer to make sure the voltage regulator stays in the temperature range of 20℃ to 27℃.	Verified

Table 8. Requirements and Verification Table for Charge Pump with LDO

Requirements	**Verification**	**Verification Status**
1. Step up to 5V+/- 5% from a 3.5-4.2V source.	1. Measure output voltage using an oscilloscope to ensure that the output voltage stays within 5% of 5V	Verified
3. Maintain a temperature range of 20℃ to 27℃.	3. Use the IR thermometer to make sure the voltage regulator stays in the temperature range of 20℃ to 27℃.	Verified

Table 9. Requirements and Verification Table for Logic Unit

Requirements	**Verification**	**Verification Status**
1. Microcontroller must successfully boot under battery power	1. a. Attempt to flash firmware over SWDIO/SWCLK using ST Link v2.1 peripheral	Verified

2. Microcontroller must communicate with ADC	2. a. Send "start stream" instruction to ADC using SPI controller b. Observe test data stream and confirm device functionality c. Perform test without ADC populated to confirm SPI communication	Verified
3. Microcontroller must be able to write to SD FAT32 filesystem	3. a. Using a computer, wipe all the contents and filesystems off of a microSD card b. Insert the microSD card into the carriage on the board. c. Flash firmware that implements creating the FAT16 filesystem on the microSD card d. Determine if microSD card has new FAT16 partition created from device	Verified
4. Microcontroller must be able to send a "begin recording" alert to our camera peripheral	4. a. Create a microSD card formatted with a Raspbian distribution. Write firmware for our MCU that will loop over and send data to the UART serial terminal pins we share with the Raspberry Pi	Unverified

	b. Connect 3.3V DC power supply to battery terminals present on board, and flash our MCU with our test firmware. c. Connect leads to expose Raspberry Pi's serial terminal and establish a connection with our test PC. d. Verify Raspberry Pi has successfully booted via our serial terminal. Using a script, verify that the other serial tty present on the Pi is receiving our dummy MCU data being sent.	
5. Microcontroller must be able to raise an alarm depending on ADC voltage detecting a seizure-like activity or a bad contact	5. a. Flash firmware to the device that can interpret all of our ADC leads' data and send printf() statements to console regarding our seizure criteria or a bad contact b. Connect 3.3V DC power supply to battery terminals present on board, and flash our MCU with our test firmware. c. Send test data that determine edge cases of our criteria down our EEG leads. Determine that our intended detection behavior is	Unverified

	present in our serial output.	

Table 10. Camera Module R&V Table

Requirements	**Verification**	**Verification Status**
1. Must have a minimum 60 fps and 240p resolution.	1. Write a device script to record and save a short video from the camera, saving it to onboard SD. Determine the fidelity of the video using software playback in VLC.	Verified
2. Be able to track both eye movements and arm movements to shoulder height from the camera footage (verify that it is mounted at a reasonable angle on the cap visor).	2. a. Mount the camera on the cap visor at the desired angle. b. Use the microcontroller to turn on the camera and collect footage in order to process and save the final camera footage onto the SD card. c. Verify using the saved camera footage that both eye movements and arm moments to shoulder height can be observed from the camera footage. d. Repeat steps A, B, and C if the verification fails.	Verified
3. Camera size is less than 40x40 mm and weight less than 30g.	3. a. Measure the length and width of the camera chip to ensure that it is less than 40x40 mm. b. Weigh the camera chip to ensure that it is less than 30g.	Verified

4. Camera footage is stored on the onboard SD card with the correct '.h264' file extension in the appropriate folder.	4. a. Run the python script for the camera module to record an output test file. b. Confirm that the output test file is uncorrupted and has a '.h264' file extension and is located in the Videos folder.	Verified

Appendix B - Software Flowchart

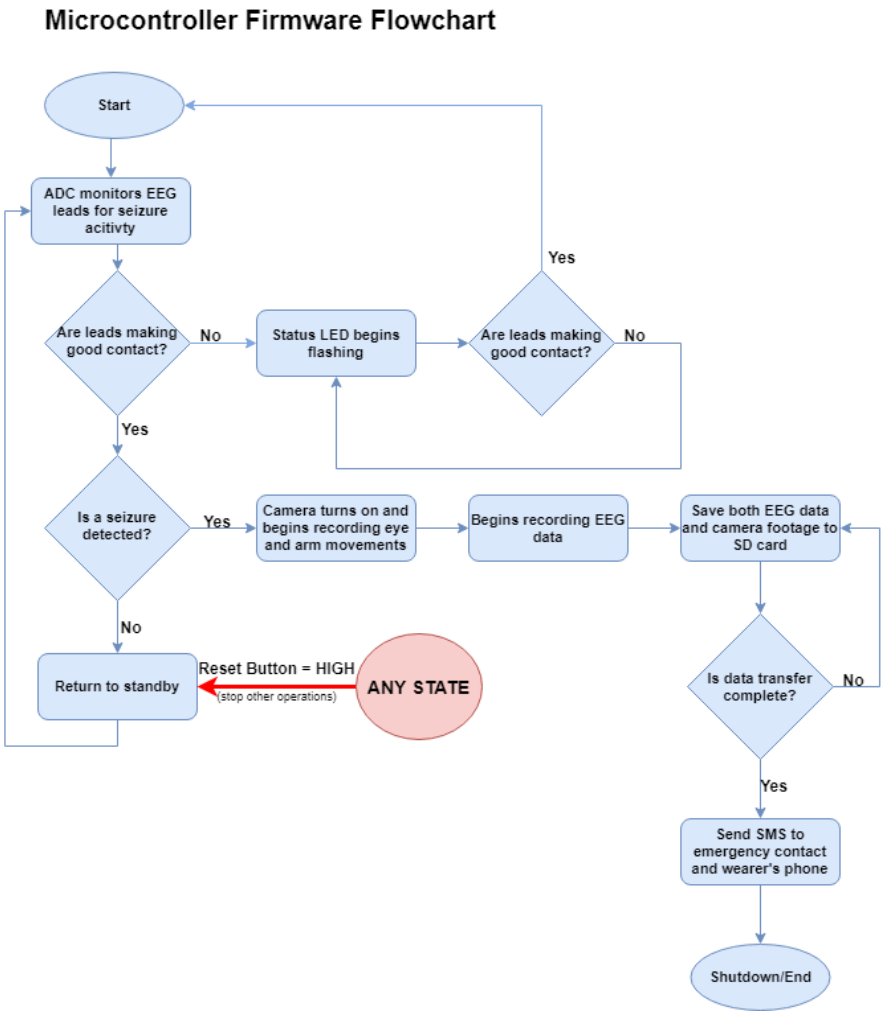

Figure 4. Microcontroller Firmware Flowchart

Appendix C - Firmware/Software Code

The code below shows how the program receives the EEG data and how the program stores these data into the SD card. As you can see there is a DefaultCallback function, which is called every time the ADS1299 is initialized. This attached registered client, DefaultCallback, will receive the EEG data from ADS1299, and process them into a ring buffer. Every time the ring buffer is full, these data will be added together. The buffer is cleared and stores the next 80 numbers. After we have 2 sums, we will compare them to see if there is a large difference. If there is a large difference, we will store these data into a txt file named with year-month-day-hour-minute-second.

```
def DefaultCallback(data):
    print (repr(data))
    global prev
    global cur
    global flag
    global buf
    if flag == 0:
      for i in range(len(data)):
        buf.append(data[i])
        prev = prev + data[i]
      if (buf.get())[0] != None:
        flag = 1
    (buf.get())[0] = 0
    if flag == 1:
      for i in range(len(data)):
        buf.append(data[i])
        cur = cur + data[i]

      if (buf.get())[0] != 0:
        prev = cur
        cur = 0
    if (cur/80) >= 1.5*(prev/80):
      os.system('python3 Record_test.py')
file_path = 'Data/'
now = datetime.now()
str_time = datetime.strftime(now,'%Y-%m-%d %H:%M:%S')
sys.stdout = open(file_path+str_time+'.txt', "w")

# init ads api
ads = ADS1299_API()

# init device
ads.openDevice()
# attach default callback
ads.registerClient(DefaultCallback)
# configure ads
ads.configure(sampling_rate=1000)
```

Figure 5. ads_test.py.

Arrays below are the result of what we receive from the ADS1299 and the same numbers closed to 0 means we do not put the electrodes on a patient's head.

```
ADS1299 API test stream starting
array([-1.21e-08, -1.21e-08, -1.21e-08, -1.21e-08, -1.21e-08, -1.21e-08,
       -1.21e-08, -1.21e-08])
array([-1.21e-08, -1.21e-08, -1.21e-08, -1.21e-08, -1.21e-08, -1.21e-08,
       -1.21e-08, -1.21e-08])
array([-1.21e-08, -1.21e-08, -1.21e-08, -1.21e-08, -1.21e-08, -1.21e-08,
       -1.21e-08, -1.21e-08])
array([-1.21e-08, -1.21e-08, -1.21e-08, -1.21e-08, -1.21e-08, -1.21e-08,
       -1.21e-08, -1.21e-08])
array([-1.21e-08, -1.21e-08, -1.21e-08, -1.21e-08, -1.21e-08, -1.21e-08,
       -1.21e-08, -1.21e-08])
array([-1.21e-08, -1.21e-08, -1.21e-08, -1.21e-08, -1.21e-08, -1.21e-08,
       -1.21e-08, -1.21e-08])
array([-1.21e-08, -1.21e-08, -1.21e-08, -1.21e-08, -1.21e-08, -1.21e-08,
       -1.21e-08, -1.21e-08])

...
ADS1299 API test stream stopping
Test Over
```

Figure 6. Test stream over SPI of ADC data.

Figure 7 shows the program that turns on and triggers the camera to record for a period of time, in this case, 20 seconds. The camera recording is written to a '.h264' file and the name is specified by the date and time the recording was taken. The camera file will be located in a folder called Videos on the Raspberry Pi Zero W's SD card. If the file is successfully saved to the Videos folder, 'New Files create' will be printed out in the command terminal (if the user is to unit test the camera module).

```
import os
from datetime import datetime
import io
import picamera
import serial
from time import sleep

secs_before = 10
trigger_fileextension = '.trg'
trigger_path = 'trigger/'
video_path = 'Videos/'

# Format of trigger file name:          2021-11-03-08-05-00.trg
# sudo apt-get install python3-serial


with picamera.PiCamera() as camera:
    camera.resolution = (300, 300) #set width and height of the image
    camera.framerate = 25 #how many image per second
    stream = io.BytesIO()
    now = datetime.now() #output the current time
    str_time = datetime.strftime(now, '%Y-%m-%d %H:%M:%S') #param1: object, param2: ex(2021-11-3 23:30:42)
    camera.start_recording(video_path+str_time+'.h264') # param1: name of the file, param2: format of the file
    camera.wait_recording(20) #start recording, recording total seconds
    print('New files create')
    camera.stop_recording() #stop recording
```

Figure **7.** Python Code for Camera Module

Figure 8 displays that the camera module successfully saves the output file to the camera module in the Videos folder. The output file also has the correct file extension and can be identified with the date and time the video was taken.

```
pi@raspberrypi:~/RaspberryPiADS1299 $ python3 Record_test.py
New files create
pi@raspberrypi:~/RaspberryPiADS1299 $ cd Videos
pi@raspberrypi:~/RaspberryPiADS1299/Videos $ ls
'2021-11-30 18:35:39.h264'  '2021-11-30 22:04:54.h264'  '2021-12-01 00:00:10.h264'  '2021-12-01 00:25:05.h264'  '2021-12-01 01:06:53.h264'
'2021-11-30 18:41:38.h264'  '2021-11-30 22:24:18.h264'  '2021-12-01 00:21:10.h264'  '2021-12-01 00:27:13.h264'  '2021-12-01 14:43:36.h264'
'2021-11-30 19:40:04.h264'  '2021-11-30 22:50:00.h264'  '2021-12-01 00:22:11.h264'  '2021-12-01 00:41:09.h264'  '2021-12-05 23:08:58.h264'
```

Figure 8. Output File Name of Video Recordings

Figure 9 displays the camera footage taken from the output file. This screenshot of the camera footage verifies that the camera module is able to capture the patient's eye movements and arm movements above shoulder height.

Figure 9. Camera Footage from the Output File