Eye-maze Venture: A project on EOG based detection and gameplay*

Sarthak Tanwani, Roshan P Mathews, and Aydin Babakhani

Electrical and Computer Engineering, University of California, Los Angeles
California, USA
sarthak73@g.ucla.edu

Abstract. This project introduces a method of EOG signal processing using a state machine requiring a low memory and processing footprint. The processed signals from the EOG waveform are interfaced with Unity to play the game "Maze Eye-venture". The game is a demonstration of using bio-signals to perform general-purpose tasks on a computer without the need for standard inputs such as a keyboard and a mouse. The gameplay is in real-time and only requires the user's eye movements to play the entire game.

Keywords: EOG Signal Processing · BioSignals · Finite State Machine

1 Introduction

Traditionally computer systems only have a fixed number of ways to take input from the user. These are the keyboard and mouse and they depend completely on the user having functioning hands and fingers. However, there are ways to read signals directly from the body that are very close to the signal generation source. There are techniques such as EMG, EEG, and EOG that, without being invasive, can detect changes in potential from within the body and these potential variations can be used as input in traditional computer systems to carry out various tasks such as opening/closing a file, controlling a wheelchair controller system or emulating a mouse cursor and a keyboard input itself. Moreover, in recent times, the concept of a computer has extended beyond traditional boundaries of a monitor, screen and a CPU such that we have complex computer systems all around us in the form of fitness watches, earbuds, smart heaters, and a lot more. In this project, we demonstrate the use of BioSignals from EOG electrodes to play a one such system in which we play the game called Maze eye venture using bioSignals from EOG electrodes. The electrodes are simply stuck on the user's body close to his eyes. This method only requires non-invasive EOG electrodes, olimex shield, and a low-cost controller with ADC and serial capabilities such as an Arduino Mega.

* Supported by organization UCLA Integrated Sensors Lab

In most cases in the industry, eye tracking is done using a dedicated camera such as [3]. Such setups require extensive memory and processing power requirement that might not be affordable for all consumers. For example, the results in [3] are obtained by using a laptop with a 1.4 GHz Intel Core i5 processor with 4GB 1600MHz DDR3 memory and a 1536 MB IntelHD Graphics card. Another example of [4] using a more cost-effective setup with a Pentium 4 2.26 GHz processor, an onboard 64 MB graphics card, and 1.25 GB RAM. On the other hand, this project uses an Arduino Mega as a processing core clocked at 16 MHz with just 256 KB of memory and 8 KB of SRAM. In addition, the power requirement is very low (of the order of milliWatts) and it can perform real-time classification by directly measuring potential changes within the eye and bypassing any extra noise sources dealt by [3] and [4].

This is distributed into the following sections. Section I describes EOG signals and its characteristics. The fundamental EOG signal processing approach is described in Section II. Section III introduces the concept of a Finite State Machine (FSM) and how it is implemented in the project. The use of timers and stopwatch is essential to limit misclassifications of eye movement which is described in section IV. Finally, the conclusions and discussion is established(/included?) in section V.

2 Electro-oculography (EOG)

Electro-oculography (EOG) is a technique used to measure eye movements and eye position. It involves recording electrical potentials generated by the eyes when they move. The eye is actually a dipole and there are different polarities of charge within the eye itself. The cornea is positively charged and the retina is negatively charged. This potential difference is called the Corneal-Retinal Potential (CRP). CRP is caused due to hyperpolarizations and depolarizations of retinal nerve cells and it ranges from 0.4 to 1 mV.

EOG works by placing electrodes on the skin near the eyes, without touching the eyes directly, to detect the electrical potential changes that occur as the eyes move. When the eyes move, the retina changes its orientation relative to the direction of the electrical field, which generates a small electrical potential. This potential can be measured using electrodes and then processed to determine the eye movement.

EOG is often used in the assessment and treatment of eye movement disorders, such as nystagmus (involuntary eye movement) and strabismus (eye misalignment). It can also be used to study the physiology of eye movement and to assess the effectiveness of treatments for these disorders.

EOG is a non-invasive and painless procedure, and it provides a high-resolution measurement of eye movements and eye position. It is often used in combination with other techniques, such as electroencephalography (EEG) and magnetic resonance imaging (MRI), to provide a comprehensive understanding of eye movements and the underlying brain activity.

3 EOG signal processing

The first part of the project is to detect and correctly classify eye movements into discrete states. The EOG electrodes near the eye can generate anywhere between 0.4 to 1 mV. However, the distinguishing characteristic of the EOG signal is not encoded in the magnitude of the response but in the shape of the signal. The EOG electrodes measure the Corneal Retinal Potential due to the positive charge infront of the cornea and a small negative charge behind the retina. While moving the eye, if the cornea comes closer to the electrodes, it would register a small positive spike at the output. Similarly, if the cornea goes further from the electrode, back to its mean position, the same electrode would register a negative spike at the output. Due to this behavior, we get the following time-series waveforms for the corresponding eye movements:

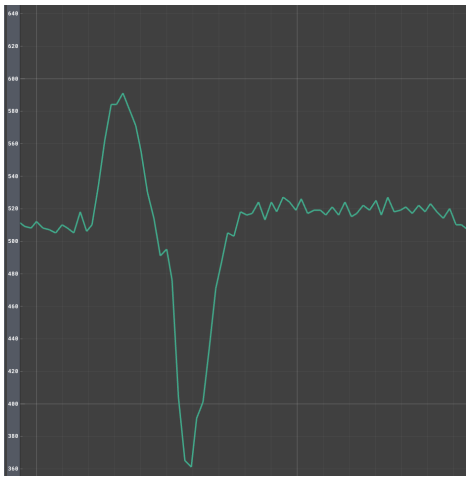

Fig. 1. Left Eye Motion

As we can see from Figs.(1, 2, 6), the left, and right eye movements have similar (and inverted) behavior that is considerably different than the blink-eye movement. As explained above, the high peak refers to the cornea moving closer to the left electrodes and the low peak refers to the cornea moving back to its mean position and moving further from the left electrode. Similarly, the right eye

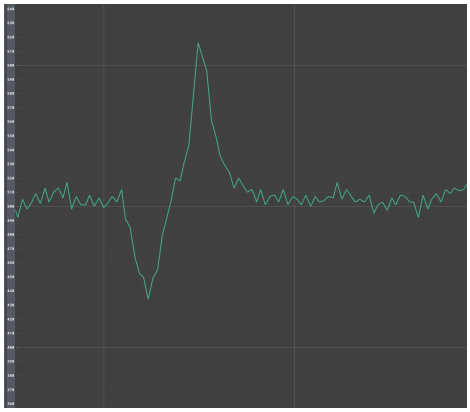

Fig. 2. Right Eye Motion

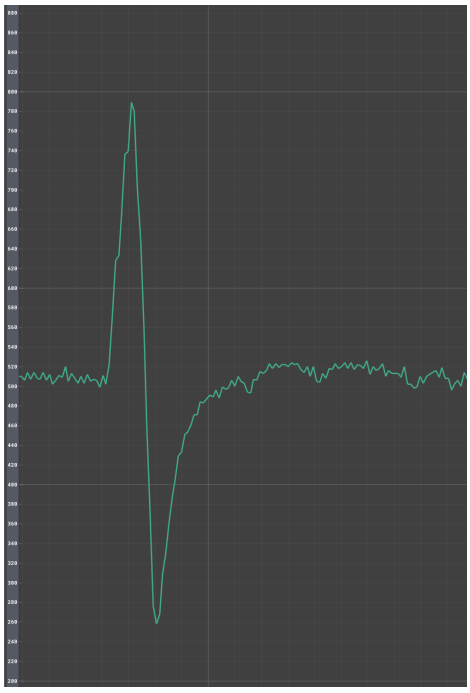

Fig. 3. Blink Eye Motion

movement has an initial low peak followed by a high peak. In order to maintain the low overhead and real-time constraints of the system, a Finite State Machine is developed. The State Machine is required since we need to identify and classify multiple peaks in a particular order as opposed to a simple threshold-based peak detection algorithm that works for a single peak.

3.1 Hardware setup and Real-time implementation

For the eye movement game, we use an Arduino Mega board connected to Olimex ECG-EKG shield that sits on top of the Arduino board. The Olimex shield mainly performs the following 2 functions:

1. The shield converts the analog differential signal (EOG bio-potentials generated by muscle), attached to its channel inputs, into a single stream of data as output.

2. The shield amplifies the signal with the total gain as the product of all the gains in each discretization stage, that is, Instrumentation Amplifier (G1 = 10), OpAmp with regulated gain (G2 = 6-101), and a 3rd order "Besselworth" filter (G3 = 3.56). By default, G2 is set to approximately ∼80. Then Gtotal = 10*(∼80)*3.56 ∼ 2848

The Arduino sends digital ADC values directly to the C# script controlling the ball object in the Unity Scene. The values are sent over UART at a rate of 1 sample every 3 ms or a frequency of ∼300 Hz. The values are sent over UART and processing of the values using the state machine is done on the application side in the C# script.

Finally, EOG values are obtained by placing the Ag/AgCl electrodes in both the horizontal and vertical of the eye.

The game works in real-time with no conceivable delays or requirement of buffer on the hardware or the C# script side. The game works smoothly mainly because of the following

4 State Machine

4.1 Introduction

A state machine is ideally a mathematical model in which a system's activity is shown as a series of states and transitions. Each state or mode that the system can function in is represented by a transition, which is a move from one state to another. State machines can be expressed graphically, like in a state diagram, or formally, like in the statechart notation.

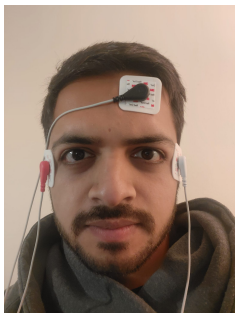

Fig. 4. Placement of electrodes for Horizontal Channel

Fig. 5. Placement of electrodes for Vertical Channel

State machines are frequently utilized in a variety of fields, including user interfaces, control systems, and embedded systems. They are particularly beneficial for modeling systems with complicated states and transitions, where the timing and order of events affect the behavior of the system.

In our system, the high and low peaks can be identified by thresholding. However, in order to capture these peaks in a particular order and classify them (high-low or low-high) we need a simultaneous mechanism that does not disrupt the continuous flow of data from the arduino to C# script. Since the data is sent at a relatively high speed (one value every 3 ms), a suitable low footprint method, both in terms of time and processing, is to implement a Finite State Machine (FSM).

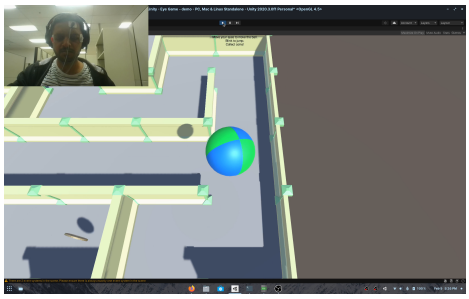

Fig. 6. Gameplay in Unity

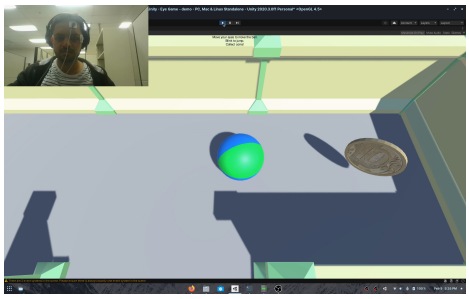

Fig. 7. Gameplay in Unity

5 State Machine

5.1 Implementation

The state machine is implemented as follows:

The state machine is implemented in a separate thread called "Read" in the C# script. This allows functions in the game to run independant of the EOG signal data processing. The Read thread starts by receiving a sample from the Serial port, that happens once every 3 ms or at a frequency of 333 Hz. Once the data from the Serial port is received and the ADC value is parsed properly, we check the magnitude of the value. A function called updateStateMachine is only called if the value crosses one of the three thresholds. We mainly need 3 different kinds of threshold to classify an eye movement as left(or up), right(or down) or blink.

Algorithm 1 Algorithm used for the State Machine Classifier

Require: $n \geq 0$
Ensure: Serial Port is connected
  baudrate ← 115200
  while *val* ← NEW SAMPLE **do**
    if *val* is greater than BLINK_THRESHOLD **then**
      state ← $BLINK$
      Stop, Reset, and Start Timer
    else if *val* is greater than BLINK_THRESHOLD but less than HIGH_THRESHOLD **then**
      if *last_state* is LOW_THRES and time since last movement is less than 500 ms **then**
        state ← $RIGHT\_MOVT$
        Stop, Reset and Start Timer
      else if *last_state* is IDLE **then**
        state ← $HIGH\_THRES$
      end if
    else if *val* is less than LOW_THRESHOLD **then**
      if *last_state* is HIGH_THRES and time since last movement is less than 500 ms **then**
        state ← $LEFT\_MOVT$
        Stop, Reset and Start Timer
      else if *last_state* is IDLE **then**
        state ← LOW_THRES
      end if
    end if
  end while

Here is the higher-level logic to identify movement1, movement2 and blink movement. In order to classify left(or up) movement, we need to identify 1

high peak followed by 1 low peak in that order. Similarly, to classify right(or down) movement, we need to identify 1 low peak followed by 1 high peak. A blink movement may cause the EOG plot shape to be similar to either right or left movement, but it has a much larger magnitude than either of them. While, classifying a left/up or right/down movement requires 2 state transitions, $IDLE \rightarrow HIGH\_THRES \rightarrow LOW\_THRES$ in the former and $IDLE \rightarrow LOW\_THRES \rightarrow HIGH\_THRES$, classifying blink movement only requires 1 state transitions. This is because, for the other two movements we need to check for both magnitude and a particular shape of the EOG waveform, whereas blink is only characterized by a very high magnitude (BLINK_THRESHOLD). Hence we have the following valid state transitions for Blink movements:

1. $IDLE \rightarrow BLINK$

2. $IDLE \rightarrow HIGH\_THRES; HIGH\_THRES \rightarrow BLINK$

Therefore, we only need to check for 3 distinct thresholds i.e. 1 high threshold, 1 low threshold and 1 blink threshold (which is numerically higher than the high threshold) to identify these movements. The implemented state machine is only updated when the ADC value crosses either of the thresholds. In any other case, the Read thread just waits for the next ADC sample to be received. This is done for a number of reasons:

1. This makes the stateMachine design simpler and some pitfalls are avoided.

2. Not calling the updateStateMachine function with each sample helps with the smooth Real-Time function of the game and saves some processing. Please note that the C# script needs to receive a sample, process it and update the state machine before receiving the next sample or within 3 ms.

Case I: Val crosses Blink Threshold i.e. val > Blink Threshold In this case, the script would check for the current state. If the current state is $IDLE$ or $HIGH\_THRES$, it will be classified as Blink motion. Please note that $LOW\_THRES \rightarrow BLINK$ is not a valid state transition as a continuous EOG waveform would have to first transition to IDLE from LOW_THRES before the state transition happens to BLINK state.

Case II: Val crosses High Threshold i.e. val > High Threshold This is the case in which the current ADC value is greater than $HIGH\_THRESHOLD$ but less than $BLINK\_THRESHOLD$ since the check for $BLINK\_THRESHOLD$ happens before this. When the val crosses $HIGH\_THRES$, it would either be the first peak or the second peak of the entire eye movement. This would be distinguished by checking for the current state. If the current state is IDLE, or if the state transition is from $IDLE \rightarrow HIGH\_THRES$, then it is the first peak and indicates the start of left/up movement. So, the new state will remain

$HIGH\_THRES$. Similarly, State transition of $LOW\_THRES \rightarrow HIGH\_THRES$ signifies a second peak or the completion of right/down eye movement. In this case, the movement is identified and new state is set to $RIGHT\_MOVT$. *Add image of both cases*

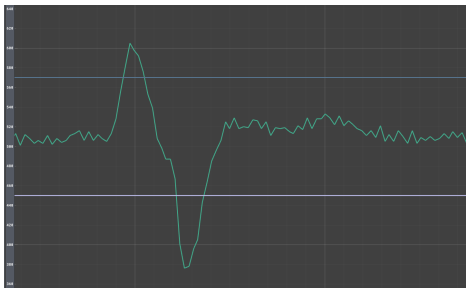

Fig. 8. Left motion with High and Low Thresholds

Case III: Val crosses Low Threshold i.e. val < Low Threshold This case is very similar to case II. If the current state is IDLE, or if the state transition is from $IDLE \rightarrow LOW\_THRES$, then it is the first peak and indicates the start of right/down movement. So, the new state will remain LOW_THRES. Similarly, State transition of $HIGH\_THRES \rightarrow LOW\_THRES$ signifies a second peak or the completion of left/up eye movement. In this case, the movement is identified and new state is set to $LEFT\_MOVT$. *Add image of both cases*

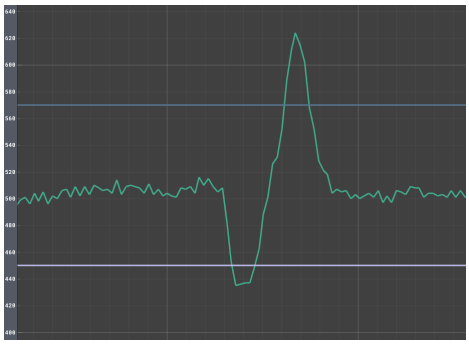

Fig. 9. Right motion with High and Low Thresholds

The above cases would either lead to an eye movement being classified ($LEFT\_MOVT/UP\_MOVT$, $RIGH\_MOVT/DOWN\_MOVT$ or BLINK) or

a state transition to an intermediate state with no complete eye movement(LOW_THRES or $HIGH\_THRES$). Intermediate states are cases when the eye is moved completely in one direction and is not back to its neutral position yet. In case of proper eye movement, the intermediate state should be temporary and only last for a few hundreds of milliseconds. Hence, we use a 1000 ms stopwatch (from the Stopwatch class in C# System.Diagnostics) in the C# script to reset the state to IDLE if the current state is an intermediate state for 2000 ms or more. There are 2 timers in the project, and their implemetation is explained in the following section.

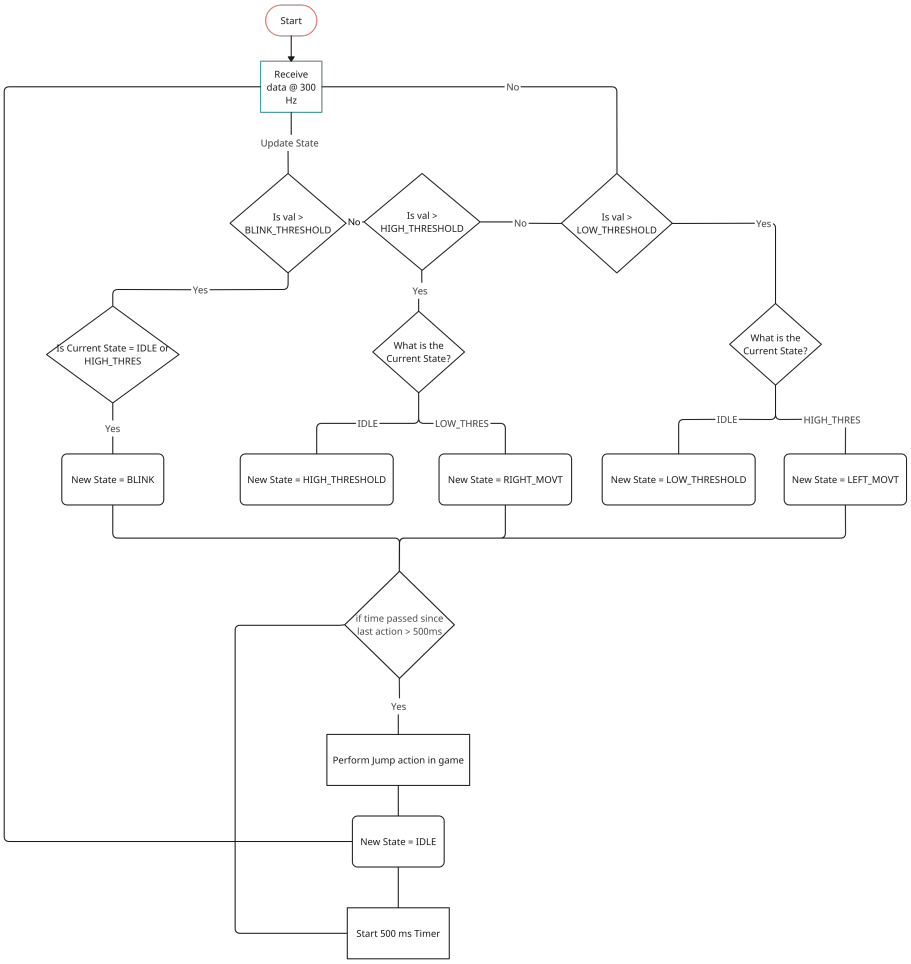

Fig. 10. State Machine Diagram

6 Stopwatch

Two different stopwatches are used in the script to eliminate specific cases of redundant eye movements.

1. Reset Stopwatch

This Stopwatch is used to reset the state from an intermediate state back to IDLE after a certain amount of time has passed and the intermediate state did not result in an eye movement in a fixed amount of time. As referred in the previous section, an intermediate(HIGH_THRES or LOW_THRES) state is when the eye moves to one of the extreme position (left, right, up or down) and has not moved back to the neutral position yet. We assume that the user would bring the cornea back to the neutral position in maximum timeframe. If the gameObject is in intermediate state for longer than this max timeframe (which is set to 2000 ms for now), the eye movement is rejected as an anomaly and the state is reset to IDLE. This stopwatch is started whenever the ADC value crosses a threshold and is check and reset with each new ADC sample.

2. Next Movement Stopwatch

Some eye movements, such as blink, require only a single state transition and hence can be identified multiple times in a very small timeframe. Also, the sphere gameObject takes some time to move completely once a movement(left, right or blink) has been identified. Therefore, a 500 ms stopWatch has been implemented that checks the time since last movement before implementing the actions of the current eye movement. The Stopwatch starts after every movement and is checked and reset at every iteration of the main thread.

7 Conclusion and Future Work

A simple implementation of a state machine was used to classify ADC data from EOG signals into left, right, up, down and blink movement. Raw EOG data was sent to Unity script by the Arduino Mega and Olimex shield via the UART protocol. Once the classification is done on the C# script, it controls the movement of the ball in the unity game with the objective of collecting all the coins in a maze. This is a low-cost, non-intrusive, demonstration of gameplay using BioSignals from the EOG and does not depend on the user to be able to control a mouse and a keyboard. Gameplay only depends on the movement of the eyes. This ensure that even people who have completely lost motor control can perform complex tasks on a computer without the help of a caretaker.

Currently, we were using an Arduino Mega(ATMega2560) running at 16MHz and has a single ADC with different channels. In future, more real-time nature can be achieved by using a dedicated ADC for each of the horizontal and vertical EOG channels. Additionally, using a Microcontroller with higher storage and

processing speed (such as an ESP32 SoC running at 80MHz) would ensure more real-time nature of the control and can provide additional processing if needed with little to no additional cost.

References

1. GitHub Repo for the project with demo video, https://github.com/sarthaktanwani/Eye-maze-venture.
2. Bott NT, Lange A, Rentz D, Buffalo E, Clopton P and Zola S (2017) Web Camera Based Eye Tracking to Assess Visual Memory on a Visual Paired Comparison Task. Front. Neurosci. 11:370. https://doi.org/10.3389/fnins.2017.00370
3. Mehrubeoglu, Mehrube Pham, Linh Le, Hung Muddu, Ramchander Ryu, Dongseok. (2011). Real-time eye tracking using a smart camera. Proceedings - Applied Imagery Pattern Recognition Workshop. 1-7. https://doi.org/10.1109/AIPR.2011.6176373.
4. S. Yathunanthan, L. U. R. Chandrasena, A. Umakanthan, V. Vasuki and S. R. Munasinghe, "Controlling a Wheelchair by Use of EOG Signal," 2008 4th International Conference on Information and Automation for Sustainability, Colombo, Sri Lanka, 2008, pp. 283-288, https://doi.org/10.1109/ICIAFS.2008.4783987.
5. M. Merino, O. Rivera, I. Gómez, A. Molina and E. Dorronzoro, "A Method of EOG Signal Processing to Detect the Direction of Eye Movements," 2010 First International Conference on Sensor Device Technologies and Applications, Venice, Italy, 2010, pp. 100-105, https://doi.org/10.1109/SENSORDEVICES.2010.25.
6. N. M. M. Noor and M. A. M. Mustafa, "Eye movement activity that affected the eye signals using electrooculography (EOG) technique," 2016 6th IEEE International Conference on Control System, Computing and Engineering (ICCSCE), Penang, Malaysia, 2016, pp. 91-95, https://doi.org/10.1109/ICCSCE.2016.7893551.
7. H. S. Dhillon, R. Singla, N. S. Rekhi and R. Jha, "EOG and EMG based virtual keyboard: A brain-computer interface," 2009 2nd IEEE International Conference on Computer Science and Information Technology, Beijing, China, 2009, pp. 259-262, https://doi.org/10.1109/ICCSIT.2009.5234951.
8. R. P. Mathews, H. Jafari Sharemi, I. Habibagahi, J. Jang, A. Ray and A. Babakhani, "Towards a Miniaturized, Low Power, Batteryless, and Wireless Bio-Potential Sensing Node," 2022 IEEE Biomedical Circuits and Systems Conference (BioCAS), Taipei, Taiwan, 2022, pp. 404-408, https://doi.org/10.1109/BioCAS54905.2022.9948685.
9. B. Estrany, P. Fuster, A. Garcia, and Y. Luo. 2009. EOG signal processing and analysis for controlling computer by eye movements. In Proceedings of the 2nd International Conference on PErvasive Technologies Related to Assistive Environments (PETRA '09). Association for Computing Machinery, New York, NY, USA, Article 18, 1–4. https://doi.org/10.1145/1579114.1579132.
10. State Machine Introduction, https://www.linkedin.com/pulse/case-state-machines-better-efficiency-predictability-lance.