Vital lab final project

Electro-oculography (EOG)

Mahdi Reza Heidari Dastjerdi 810100273 idhamazer200281@gmail.com
Mohammad Amin Kolahdoozan 810100257 m.aminkolah10@gmail.com

Objective:

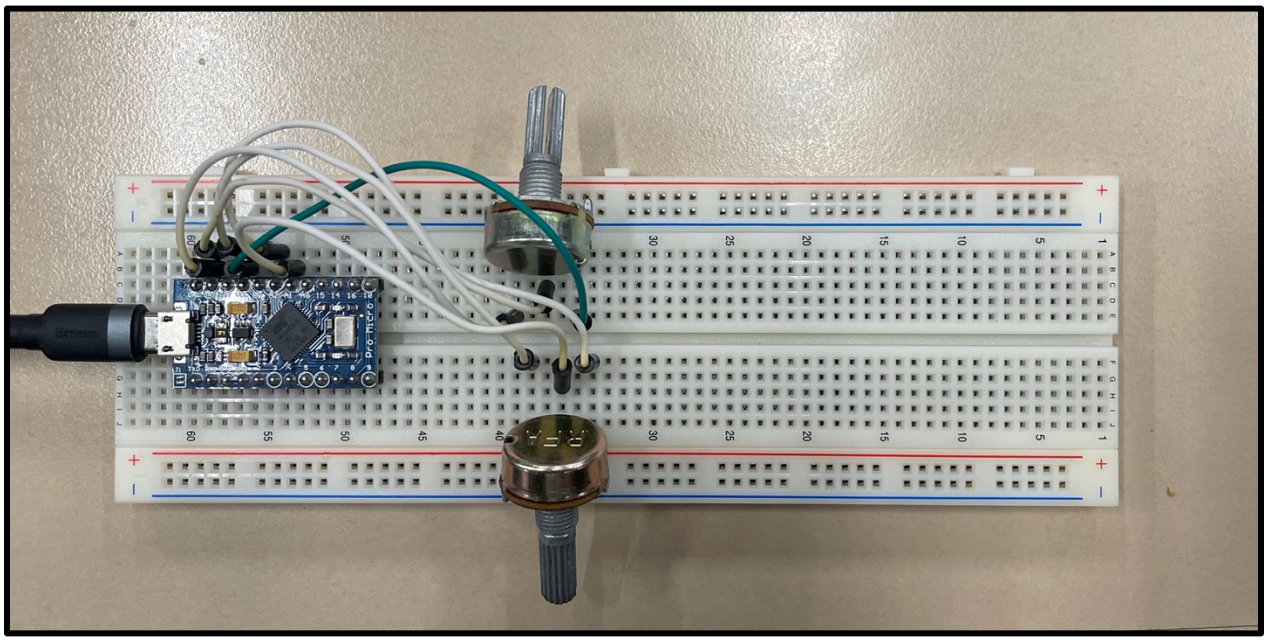
In the first or second week of the experiment, you became familiar with the EOG signal. You observed that with this signal, it is approximately possible to guess where a person is looking. In this project, we aim to take the output of this signal and, with some modifications, connect it to the computer as a mouse. This way, by moving the eyes, the mouse cursor will move on the screen.

Answer :

For this project, we will use an Arduino Pro Micro. With the help of this board, we will map the read signals onto the screen and move the mouse accordingly.
First, we will read the eye signals using an EOG device and amplify them to be interpretable by the board. Since the board only understands positive signals, we need to shift the range so that all voltages become positive.
The horizontal eye signal will be set to 2 millivolts, and the vertical eye signal will be set to 5 millivolts.

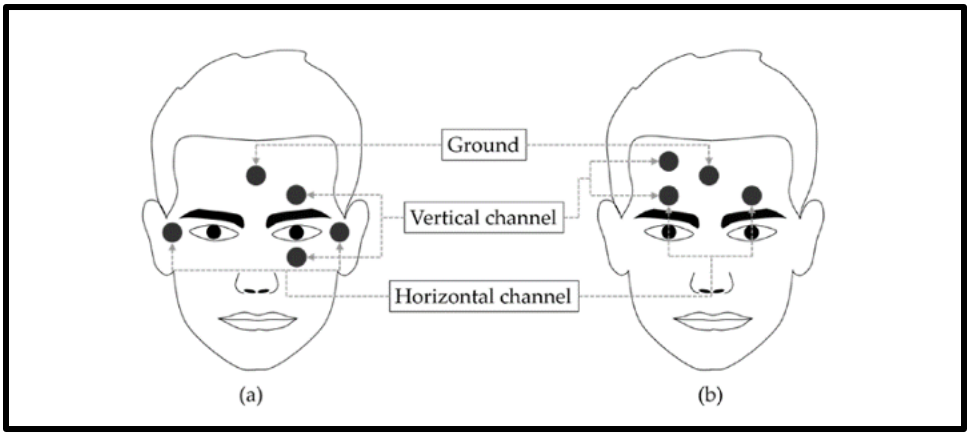
In the next step, we need to test the mouse movement using potentiometers first. Instead of using the EOG, we will use potentiometers: one for horizontal movement and one for vertical movement.
To connect the potentiometers, connect one terminal to the GND of the Arduino, another terminal to VCC, and the middle terminal to A1 or A2. This way, the read signals will be saved in these variables.

As you can see, the closed circuit is shown in the diagram below.



After this, we wrote a code using the Arduino and potentiometers to move the mouse. However, we encountered a problem: the signal was noisy. Since the EOG signal inherently has noise, it caused the mouse movement to be erratic. Therefore, we used a moving average filter. This filter collected 200 data points, calculated the average, and then sent the resulting value to the mouse. This way, the noise was reduced, and the movement became smoother.

Now let's move on to placing the electrodes. There are two methods for this, as you can see in the diagram. We opted for method A.



Optional: What is the difference between these two methods?
The difference between these two methods is that in the second method, since both electrodes are above the eye, detecting horizontal movement is more difficult. Therefore, using the first method is easier because it creates a greater voltage difference between the two ends of the eye. However, the strength of the second method lies in the fact that less noise is introduced into the signal.

The code explanation:

Include Necessary Libraries

```
#include <Mouse.h>
```

This line includes the Mouse library, which allows the Arduino to control the mouse pointer on a connected computer.

Pin Definitions and Constants

```
// Pin definitions
const int horizPin = A1;
const int vertPin = A2;

// Sensitivity to mouse movement
const int sensitivity = 1; // Adjust this value to change how fast the mouse moves

// Screen resolution (example values)
const int screenWidth = 1920;
const int screenHeight = 1080;
```

These lines define the analog pins used for horizontal and vertical inputs from a joystick or similar device. The sensitivity constant can be adjusted to change the speed of the mouse movement. screenWidth and screenHeight represent the resolution of the screen.

Arrays for Averaging Analog Readings

```
int arrayH[200];
int arrayV[200];
int index_arrayH = 0;
int index_arrayV = 0;
```

These arrays store the last 200 readings from the horizontal and vertical analog inputs to calculate a moving average, which helps smooth out the mouse movement.

Setup Function

```
void setup() {
  // Initialize serial communication for debugging (optional)
  Serial.begin(9600);

  // Initialize the Mouse library
  Mouse.begin();

  // Initialize the arrays to zero
  for (int i = 0; i < 200; i++) {
    arrayH[i] = 0;
    arrayV[i] = 0;
  }

  delay(3000); // Wait for 3 seconds before starting

  // Move the cursor to the middle of the screen initially
  Mouse.move(screenWidth / 2, screenHeight / 2, 0);
}
```

In the setup function:
Serial communication is initialized for debugging purposes.
The Mouse library is initialized.
Arrays are filled with zeroes to start.
A delay of 3 seconds is introduced to ensure everything is ready.
The cursor is moved to the middle of the screen.

```
void loop() {
  // Read the voltages from A1 and A2
  int horizReading = analogRead(horizPin);
  int vertReading = analogRead(vertPin);

  // Store the readings in the arrays
  arrayH[index_arrayH] = horizReading;
  index_arrayH = (index_arrayH + 1) % 200;
  arrayV[index_arrayV] = vertReading;
  index_arrayV = (index_arrayV + 1) % 200;

  // Calculate the moving average
  int movingavrH = 0;
  int movingavrV = 0;
  for (int i = 0; i < 200; i++) {
    movingavrH += arrayH[i];
    movingavrV += arrayV[i];
  }
  horizReading = movingavrH / 200;
  vertReading = movingavrV / 200;

  // Map the readings to screen coordinates
  int horizCoord = map(horizReading, 0, 1023, 0, screenWidth) * 3;
  int vertCoord = map(vertReading, 0, 1023, 0, screenHeight) * 3;
```

```
  // Debugging output (optional)
  Serial.print("Horiz Reading: ");
  Serial.print(horizReading);
  Serial.print(", Vert Reading: ");
  Serial.print(vertReading);
  Serial.print(" -> Horiz Coord: ");
  Serial.print(horizCoord);
  Serial.print(", Vert Coord: ");
  Serial.println(vertCoord);

  // Move the mouse to the mapped coordinates
  static int prevHorizCoord = screenWidth / 2;
  static int prevVertCoord = screenHeight / 2;

  int horizMove = horizCoord - prevHorizCoord;
  int vertMove = vertCoord - prevVertCoord;

  Mouse.move(horizMove, vertMove);

  prevHorizCoord = horizCoord;
  prevVertCoord = vertCoord;

  // Small delay to allow time for the next reading
  delay(10);
}
```

In the loop function:

Analog readings from the horizontal and vertical pins are taken.
These readings are stored in arrays, and the indices are updated circularly.
A moving average of the last 200 readings is calculated to smooth out the input values.
The averaged readings are mapped to screen coordinates. The map function scales the input readings to the screen resolution.
Debug information is printed to the serial monitor.
The mouse is moved relative to the previous position. Since the Mouse.move() function only supports relative movement, the code calculates the difference from the previous position.
A short delay is added to allow time for the next reading, making the movement smoother.

Summary

This code reads analog values from two pins, averages these readings, and uses them to control the mouse cursor on a computer screen. The cursor movement is smoothed out by averaging the readings over the last 200 samples.