BMEG 3111 Course Project

Paralyse need to type word

Stage 2 Report

Group 6

AU Wai Tak, Wales	**(1155175068)**
CHAN Cheuk Ka	**(1155174356)**
CHEUNG Ho Lun, Louis	**(1155174348)**
LAU Man Hei, Wes	**(1155163433)**
HEUNG Hoi Ying, Helen	**(1155176975)**
HO Yu On, Martin	**(1155175831)**
WONG Kin Hang, Koby	**(1155175687)**

Table of Contents

1 Brief Project Overview

Our project aims to help those who are disabled by amyotrophic lateral sclerosis and similar quadriplegia-inducing diseases to use a computer, including typing and cursor navigation. Our product consists of a glasses-mounted gyroscope for cursor movement and an electromyograph (EMG) for mouse clicks. Users can use their head movements to control the mouse cursor and contract their cheek muscles for mouse clicks.

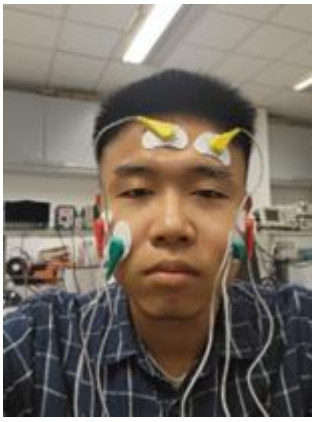

Figure 1
User with EMG electrodes attached to his facial muscles

2 Circuit Design

2.1 Circuit Overview

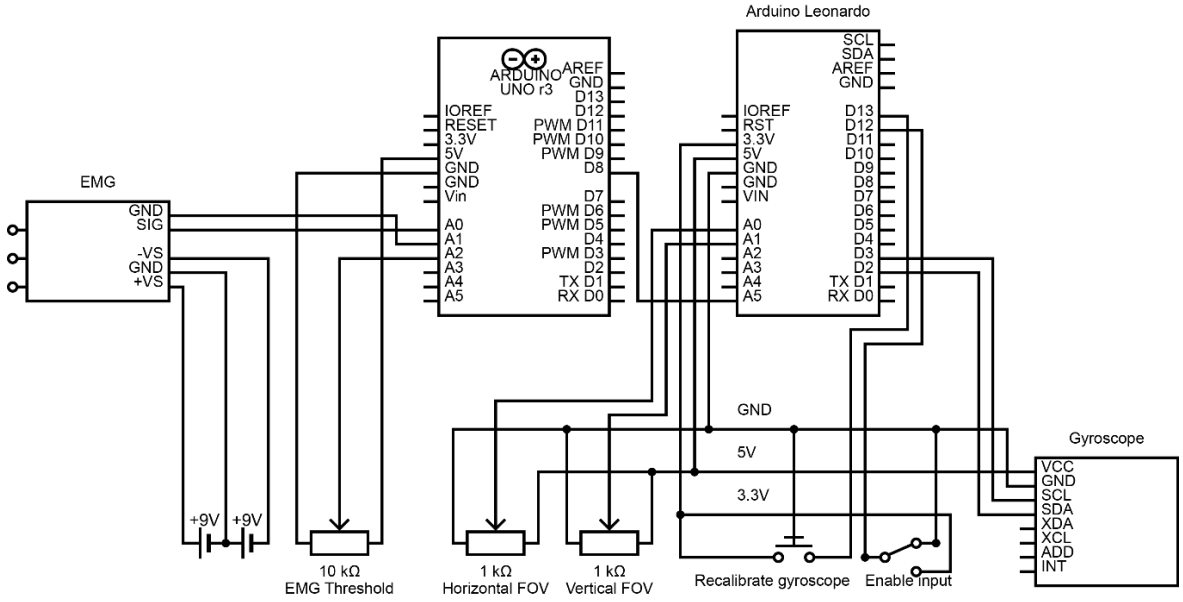

Figure 2
Circuit diagram

The project consists of an electromyograph board, a gyroscope, an Arduino Leonardo, an Arduino Uno, and various potentiometers and switches for input. Considering that many

components require access to +5V, +3.3V, and ground pins, we have built rails similar to that in a typical breadboard for ease of wiring.

There are multiple interfaces the user can use to control this system:

- An EMG
- A potentiometer to control the EMG threshold value
- A gyroscope
- Two potentiometers to control the gyroscope's horizontal and vertical field of view
- A push button to re-calibrate and re-centre the gyroscope
- A master flick switch to turn the the mouse control override on or off by the system

2.2 Electromyograph

An EMG board is utilised in this project. Two external 9V batteries are required to power the board, as instructed by the manufacturer. The signal and ground pins are connected to the Arduino Uno for processing. A potentiometer is used to control the threshold value for a positive EMG verdict.

2.3 Gyroscope

The gyroscope is connected to the Arduino Leonardo via the I^2C protocol. Two potentiometers are used to control the field of view angle the user is required to operate within in order to control the cursor. A press switch is used to re-calibrate the gyroscope and re-centre the cursor.

3 Mechanical Design

3.1 Project Box

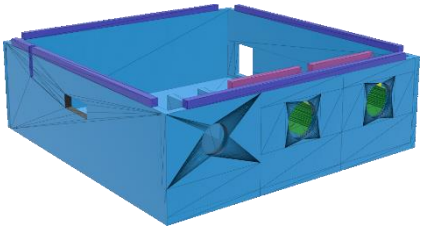

Figure 3
Project box, stage 1

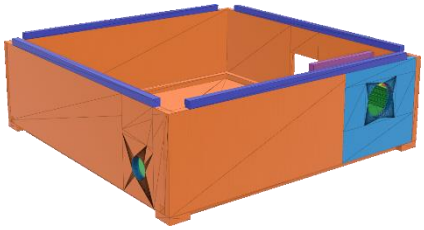

Figure 4
Project box, stage 2

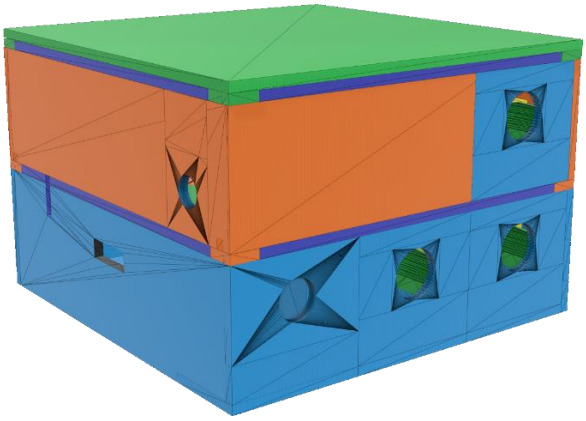

Figure 5
Project box, assembled

Our project box is split into two stages. Stage 1 houses the Arduino Leonardo and the potentiometers and switches connected to it, while stage 2 houses the Arduino Uno and the EMG board and potentiometer connected to it.

3.2 Gyroscope mount

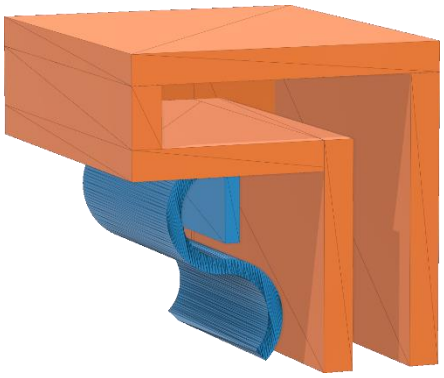

Figure 6
Gyroscope mount

To mount the gyroscope onto the user's glasses, we have designed a clip-on mount for ease of use.

4 Software Design

4.1 Software Overview

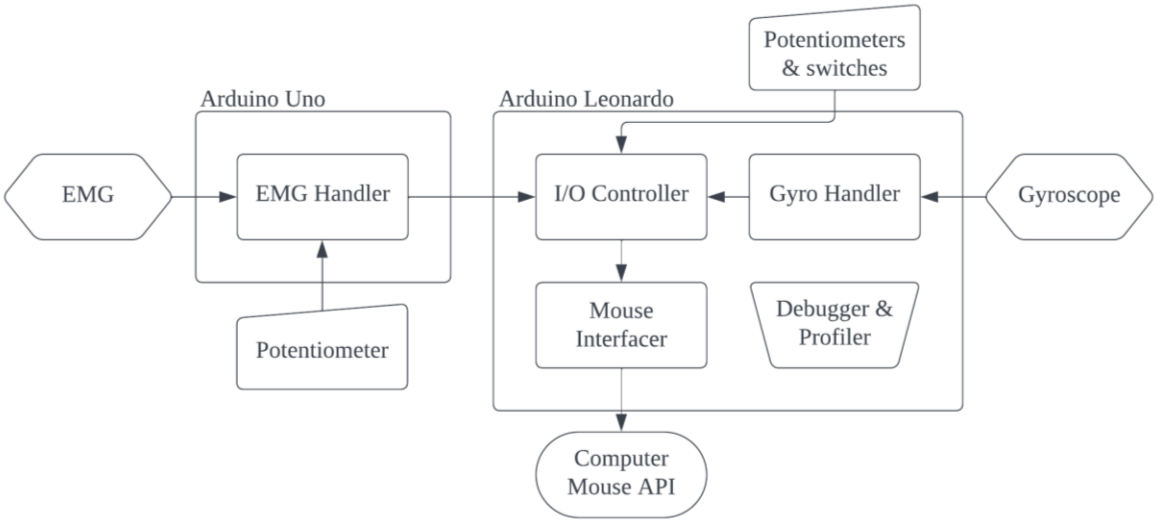

Figure 7
Software design and interface diagram

We separated the responsibilities of the program into independent blocks, each handling one independent part of the project.

4.2 EMG Interpreter

The EMG Interpreter receives signals from the EMG board and outputs an EMG verdict to the Arduino Leonardo after processing.

```
void EMGInterpreter::loop() {
    emgSignal = analogRead(EMG_SIGNAL_PIN) - analogRead(EMG_GROUND_PIN);
    emgThreshold = analogRead(THRESHOLD_POTENTIOMETER_PIN);
    Serial.println((String)emgSignal + ", " + (String)emgThreshold + ", "
 + (String)thresholdCooldownCounter);

    if (emgSignal > emgThreshold) {
        thresholdCooldownCounter += 10;
        thresholdCooldownCounter = min(thresholdCooldownCounter, thresholdCooldown);
    } else {
        thresholdCooldownCounter = max(thresholdCooldownCounter - 1, 0);
    }

    digitalWrite(EMG_OUTPUT_PIN, thresholdCooldownCounter ≥ 5 ? HIGH : LOW);
}
```

The `EMGInterpreter` collects user input from the potentiometer to adjust the threshold value and input signals from the EMG board, then checks if the EMG signal exceeds the user-defined threshold value before finally giving a verdict to the Arduino Leonardo via `EMG_OUTPUT_PIN`.

Due to the threshold calibration process requiring the use of the serial plotter to see the values more easily in curves, we have thus opted to use another Arduino board in order not to clutter the serial plotter plots for ease of discerning.

4.3 I/O Controller

The I/O Controller is the main controlling unit of this program. It handles all the input and output traffic and acts to coordinate between the different components.

```cpp
void IOController::loop() {
    manageIOStates();
    if (!mouseEnabled) {
        return;
    }
    //
    if (digitalRead(PRESS_SWITCH_PIN)) {
        gyroHandler->initialise();
    }
    gyroHandler->loop();
    setMousePosition();
}
```

```cpp
void IOController::manageIOStates() {
    // ! mouse position
    bool newMouseEnabled = digitalRead(MOUSE_ENABLE_PIN);
    if (newMouseEnabled != mouseEnabled) {
        mouseEnabled = newMouseEnabled;
        Debugger::log("IO - Mouse", mouseEnabled ? "Mouse override enabled" : "Mouse override disabled");
        if (!mouseEnabled) {
            mouseInterfacer->mouseClick(false);
        }
    }

    // ! mouse click
    if (mouseEnabled) {
        mouseInterfacer->mouseClick(analogRead(EMG_INPUT_PIN) > 800);
    }

    // ! FOV potentiometers
    static Vector2 oldFOV = Vector2(0, 0);
    FOV.x = MathLite::roundToNearest(MathLite::map(analogRead(X_FOV_PIN), 0, 1024, 0, 90), FOV_STEP);
    FOV.y = MathLite::roundToNearest(MathLite::map(analogRead(Y_FOV_PIN), 0, 1024, 0, 90), FOV_STEP);
    if (oldFOV.x != FOV.x || oldFOV.y != FOV.y) {
        oldFOV = FOV;
        Debugger::log(
            "IO - FOV",
            "(" + (String)analogRead(X_FOV_PIN) + ", " + (String)analogRead(Y_FOV_PIN) +
            ") -> (" + (String)FOV.x + ", " + (String)FOV.y + ")"
        );
    }

    return;
}
```

```cpp
void IOController::setMousePosition() {
    double normalisedYaw = MathLite::clamp(gyroHandler->getYPRAngles(0) / M_PI * 180 / FOV.x * 2, -1, 1);
    double normalisedPitch = MathLite::clamp(gyroHandler->getYPRAngles(1) / M_PI * 180 / FOV.y * 2, -1, 1);

    mouseInterfacer->mouseMove(normalisedYaw, normalisedPitch);
}
```

The `IOController` collects user input from the potentiometers to adjust the internal field of view settings, then collects input signals from the `gyroHandler` and `EMG_INPUT_PIN` before finally pushing the desired cursor position and mouse button states to the `mouseInterfacer`. The yaw of the user's head is interpreted as controlling the horizontal position of the mouse cursor, and the pitch is interpreted as the vertical position.

4.4 Mouse Interfacer

```
void MouseInterfacer::mouseMove(double x, double y) {  //? use normalised coordinates [-1, 1]
    AbsMouse.move(MathLite::map(x, -1, 1, 0, SCREEN_X), MathLite::map(-y, -1, 1, 0, SCREEN_Y));
}
```

```
void MouseInterfacer::mouseClick(bool newIsLeftClicking) {
    if (isLeftClicking != newIsLeftClicking) {
        isLeftClicking = newIsLeftClicking;
        if (isLeftClicking) {
            AbsMouse.press(MOUSE_LEFT);
        } else {
            AbsMouse.release(MOUSE_LEFT);
        }
        // Debugger::log("Mouse", isLeftClicking ? "Left Clicked" : "Left Click Released");
        // Debugger::log("Mouse", analogRead(A5) > 800 ? "HIGH" : "LOW");
    }
}
```

To control the mouse cursor, we used an external library `<AbsMouse.h>` since it allows us to control the absolute position of the mouse cursor instead of only its deltas.
Note that the implementation of `mouseClick()` allows the left mouse button to be held down instead of only clicking, thus enabling dragging inputs to be performed by the user.

4.5 Supplementary Libraries

We have written two helper libraries to be used in other parts of the program.

```
class MathLite {
   public:
    static double map(double value, double fromLow, double fromHigh, double toLow, double toHigh);
    static double clamp(double value, double min, double max);
    static double absolute(double value);
    static double roundToNearest(double value, double nearest);
};
```

```
class Vector2 {
   public:
    double x;
    double y;

    Vector2(double x, double y) {
        this->x = x;
        this->y = y;
    }

    double getLength() { return sqrt(x * x + y * y); }
    double getDistance(Vector2 other) { return sqrt(pow(x - other.x, 2) + pow(y - other.y, 2)); }
    static double getDistance(Vector2 a, Vector2 b) { return a.getDistance(b); }
    static Vector2 lerpPoints(Vector2 a, Vector2 b, double t);
};
```

5 Appendix

Below are pictures of the work-in-progress circuits and wirings.

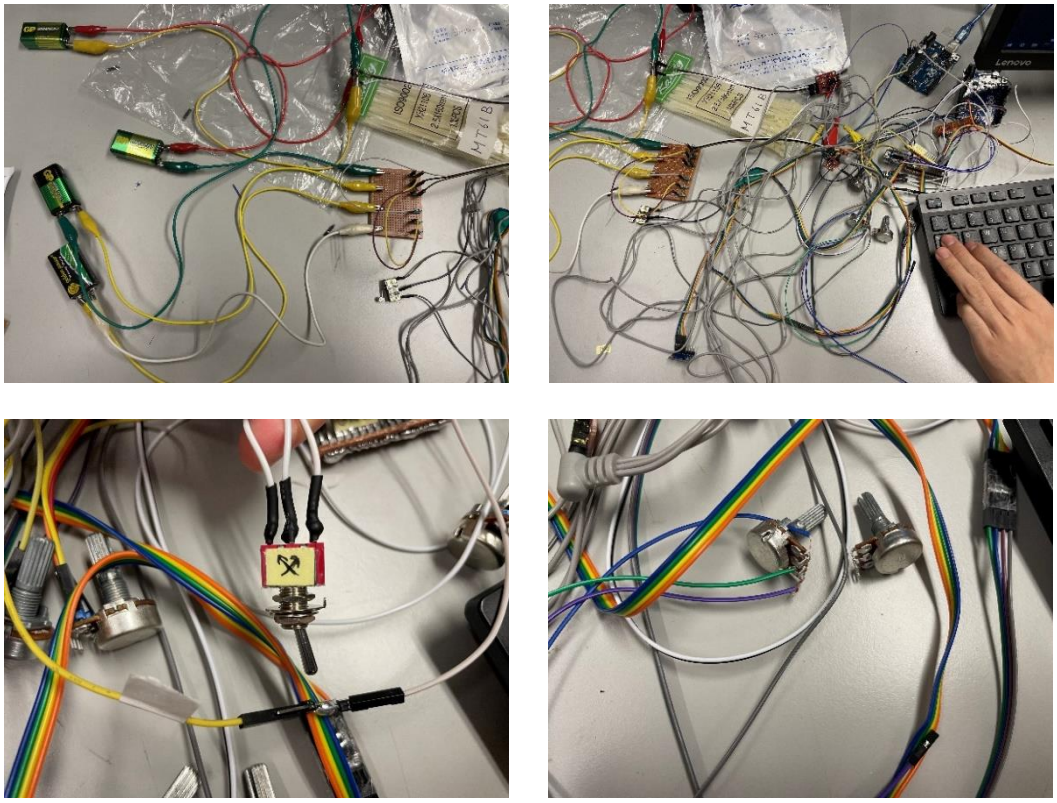

Figure 8
Circuits and wirings at different stages