BMEG 3111 Course Project

Paralyse need to type word

Stage 3 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

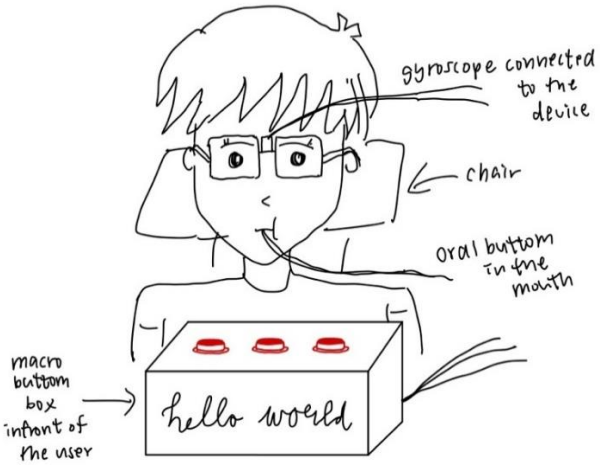


Figure 1
User of the "Paralyse need to type word ™" system, artist's rendition

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 oral push button for left mouse clicks. Users can use their head movements to control the mouse cursor and their tongue to push the push button in their mouth for mouse clicks.

2 Circuit Design

2.1 Circuit Overview

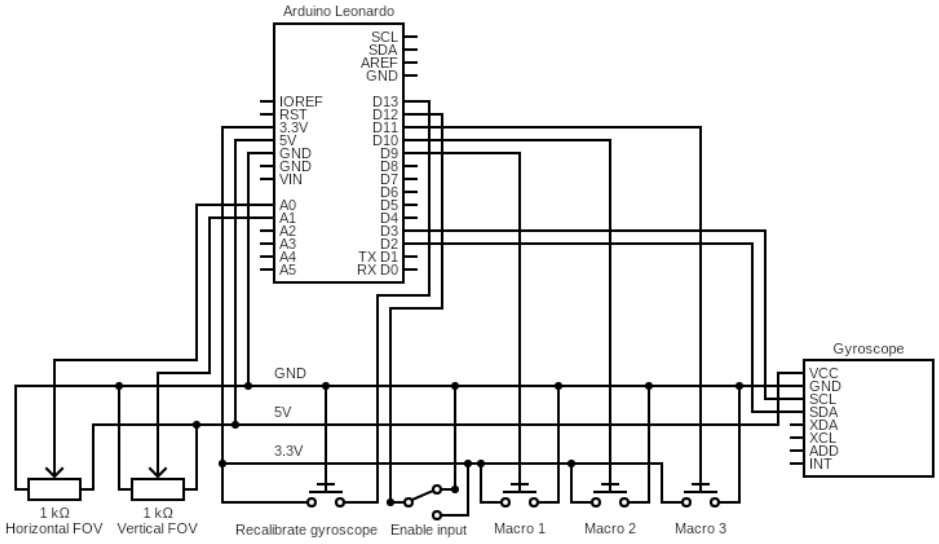


Figure 2
Circuit diagram

The project consists of an oral button, a gyroscope, three macro buttons, an Arduino Leonardo, 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:

- A gyroscope (MPU-6050)
- Two potentiometers to control the gyroscope's horizontal and vertical field of view
- An oral button to initiate a left mouse click
- Three macro buttons to initiate custom macro actions
- A push button to re-calibrate and re-centre the gyroscope
- A master flick switch to turn the mouse control override on or off by the system

2.2 Oral Button

An oral button is used to initiate a left mouse click; it can be pressed and held to drag. The user can use their tongue to press the button.

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 must operate within to control the cursor. A press switch is used to re-calibrate the gyroscope and re-centre the cursor.

2.4 Macro Buttons

Three macro buttons are used to initiate three customisable macro actions.

3 Mechanical Design

3.1 Project Box

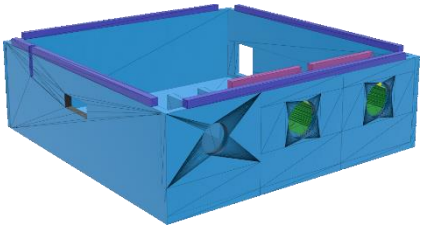


Figure 3
Project box blueprint, stage 1

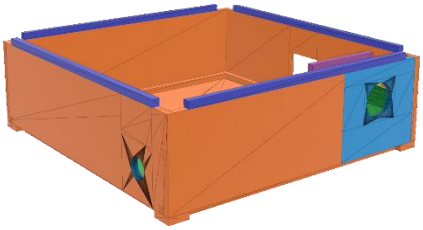


Figure 4
Project box blueprint, stage 2

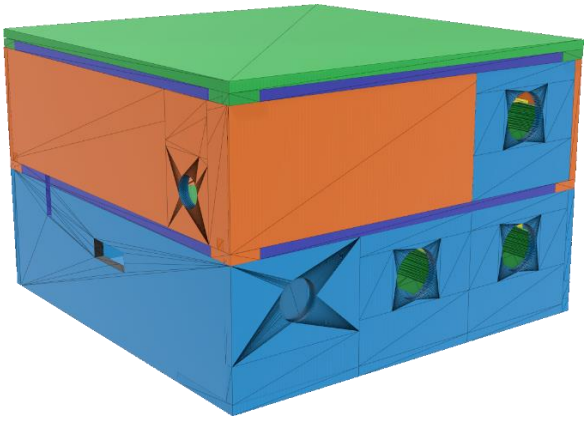


Figure 5
Project box blueprint, assembled

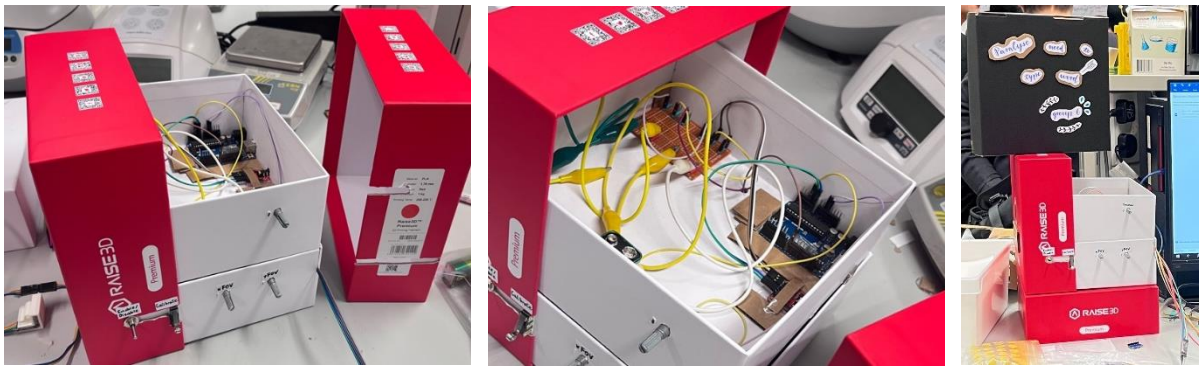


Figure 6
Project box, assembled

Figure 7
Potentiometers for various settings

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 wires and rails of the system and outlets for routing outgoing wires.

3.2 Gyroscope Mount

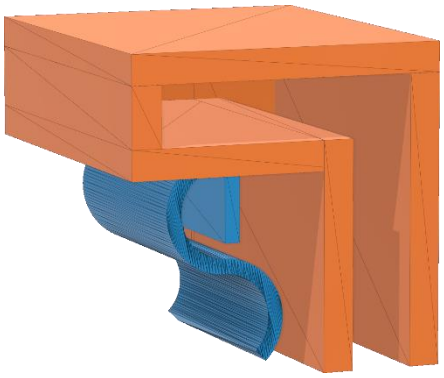


Figure 8
Gyroscope mount blueprint

We have designed a clip-on mount for ease of use to mount the gyroscope onto the bridge of the user's glasses.

3.3 Macro Button Box

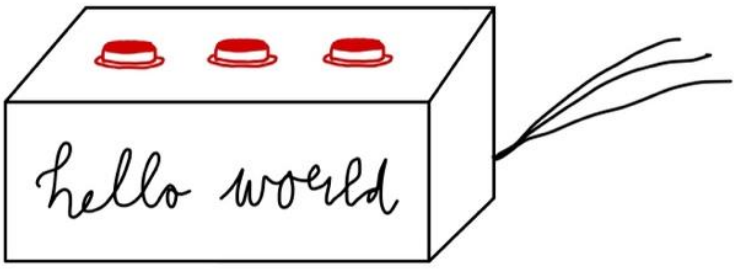


Figure 9
Macro button box, artist's rendition

Figure 10
Macro button box blueprint

We have designed a simple macro button box to house the macro buttons to be strapped onto the user's chest.

4 Software Design

4.1 Software Overview

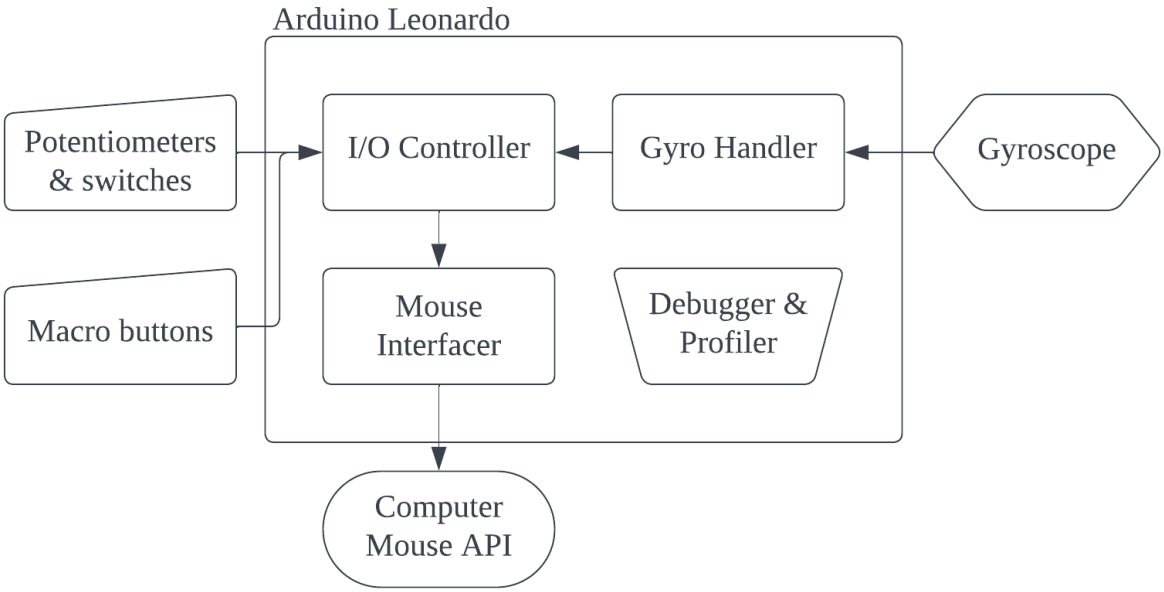


Figure 11
Software design and interface diagram

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

4.2 I/O Controller

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

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

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

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

    // ! 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 + ")"
        );
    }

    // ! macros
    if (inputEnabled) {
        // detect rising edge
        if (digitalRead(MACRO_1_PIN) && !macro1Pressed) {
            macro1Pressed = true;
            doMacro(1);
            return;
        }
        if (digitalRead(MACRO_2_PIN) && !macro2Pressed) {
            macro2Pressed = true;
            doMacro(2);
            return;
        }
        if (digitalRead(MACRO_3_PIN) && !macro3Pressed) {
            macro3Pressed = true;
            doMacro(3);
            return;
        }
    }
    if (!digitalRead(MACRO_1_PIN) && macro1Pressed) {
        macro1Pressed = false;
    }
    if (!digitalRead(MACRO_2_PIN) && macro2Pressed) {
        macro2Pressed = false;
    }
    if (!digitalRead(MACRO_3_PIN) && macro3Pressed) {
        macro3Pressed = false;
    }

    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);
}
```

```cpp
void IOController::doMacro(int macroID) {
    switch (macroID) {
        case 1:
            Debugger::log("IO - Macro", "1: Ctrl + Z pressed");
            pressControl;
            pressZ;
            break;
        case 2:
            Debugger::log("IO - Macro", "2: Calibrate pressed");
            gyroHandler->initialise();
            break;
        case 3:
            Debugger::log("IO - Macro", "3: Ctrl + Backspace pressed");
            pressControl;
            pressBackspace;
            break;
        default:
            Debugger::log("IO - Macro", "Macro ID not found");
            break;
    }
    return;
}
/* ------------------------------------------------------------------------ */
#define pressControl                                                \
    {                                                               \
        mouseInterfacer->mouseMove(0.3 * 2 - 1, 0.04 * 2 - 1); \
        mouseInterfacer->mouseClick(true);                          \
        delay(10);                                                  \
        mouseInterfacer->mouseClick(false);                         \
        delay(10);                                                  \
    }
#define pressZ                                                      \
    {                                                               \
        mouseInterfacer->mouseMove(0.32 * 2 - 1, 0.07 * 2 - 1); \
        mouseInterfacer->mouseClick(true);                          \
        delay(10);                                                  \
        mouseInterfacer->mouseClick(false);                         \
        delay(10);                                                  \
    }
#define pressBackspace                                              \
    {                                                               \
        mouseInterfacer->mouseMove(0.75 * 2 - 1, 0.25 * 2 - 1); \
        mouseInterfacer->mouseClick(true);                          \
        delay(10);                                                  \
        mouseInterfacer->mouseClick(false);                         \
        delay(10);                                                  \
    }
```

The `IOController` collects user input from the potentiometers to adjust the internal field of view (FOV) settings, then gathers input signals from the `gyroHandler` 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.

The `IOController` also monitors signals from the macro buttons and detects rising edges to initiate the corresponding macro actions. The default macro actions are set as undo (Control + Z), calibrate gyroscope, and delete word (Control + Backspace); however, the macros are designed to be easily changed and extended.

4.3 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.

4.4 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 Product Advantages

5.1 Shallow Learning Curve

Since our product only requires our users to aim the cursor in the same manner as a standard mouse, it is intuitive and has a minimal learning curve. Anyone who has prior experience with a regular mouse should be able to pick up our input method quickly and intuitively. It has a very shallow learning curve and a low skill floor, drastically increasing its accessibility.

5.2 High Speed and Consistency

The ability to absolutely position the mouse cursor with the gyroscope instead of controlling the mouse cursor deltas dramatically enhances the speed and stability of the system. Since the speed of the cursor is controlled entirely by the users' motions instead of a predetermined speed value, it has a high consistency, which allows users to acquire muscle memory as they use the system to enhance their typing speed further in the future.

From our testing, we can easily achieve 12 words per minute.

5.3 High Customisability and Extensibility

The default setup includes three macro buttons for undo (Control + Z), calibrate gyroscope, and delete word (Control + Backspace. Users can change the function of individual macro buttons to fit their needs. For example, users may change the function of undo to copy (Control + C) and the function of deleting words to paste (Control + V) or even construct a long chain of inputs that can perform a complex task like opening their email app and logging in. Users can also extend the functionality by adding more buttons to the box to customise their setup.

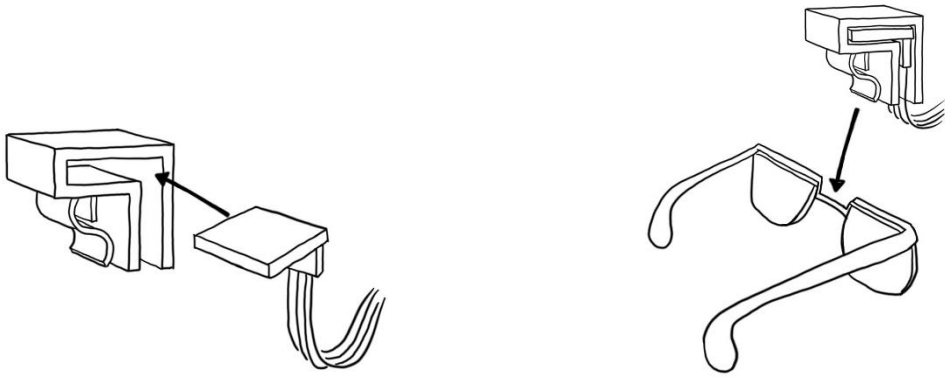
6 Measurement Procedures

Our product uses a gyroscope to sense the user's head movement based on its functionality of measuring angular velocity and rotational motion. The gyroscope we used is an MPU-6050, a microelectromechanical motion-tracking device. Gyroscopes operate based on the principle of the conservation of angular momentum. The working principle of a gyroscope involves utilising the inertia of a spinning rotor to resist changes in its orientation [1]. In our case, two axes of angular tracking is enough to coordinate a virtual keyboard. The gyroscope can detect orientation changes and provide information about the rate and direction of rotation.

7 User Manual

7.1 Setup Instructions

7.1.1 Mounting the gyroscope



1. Insert the gyroscope PCB into the gyroscope mount.
2. Attach the gyroscope mount securely to the bridge of the user's glasses.
3. Adjust the position of the gyroscope to ensure it aligns with your head movements.

7.1.2 Wearing the macro button box

1. Secure the macro button box in front of the user's chest using a strap.
2. Adjust the position of the box until it is high enough such that the user can press the buttons with their chin but low enough such that regular typing movements will not lead to accidental inputs of the buttons.

7.1.3 Placing the oral button into the user's mouth

1. Wrap the mouse button with plastic wrap. Make sure the seal is watertight.
2. Use rubbing alcohol to clean the wrap surface. Allow it to dry.
3. Insert the button into the user's mouth so that the user can push it with their tongue. Place also a piece of cotton alongside to absorb excess saliva if necessary.

7.2 Basic Operation Instructions

7.2.1 Enabling the system

1. Flip the mouse control override switch to the "on" position to turn it on. You can flip the override off anytime to temporarily disable the input system.
2. Upon turning on initially, the gyroscope will automatically calibrate itself. During calibration, the user should hold their head still in the most comfortable neutral position until the calibration is over, which is indicated by the mouse cursor reappearing at the centre of the screen. Future user-induced calibrations are to be performed in the same manner.

7.2.2 Moving the mouse cursor

1. Rotate your head in accordance with the desired direction of the mouse location. The yaw of your head controls the horizontal position of the mouse cursor, while the pitch of your head controls the vertical position of the mouse cursor.
2. *(Initial setup only)* Adjust the horizontal and vertical FOV using the potentiometers until the cursor movement best matches the user's head movement.

7.2.3 Clicking the left mouse button

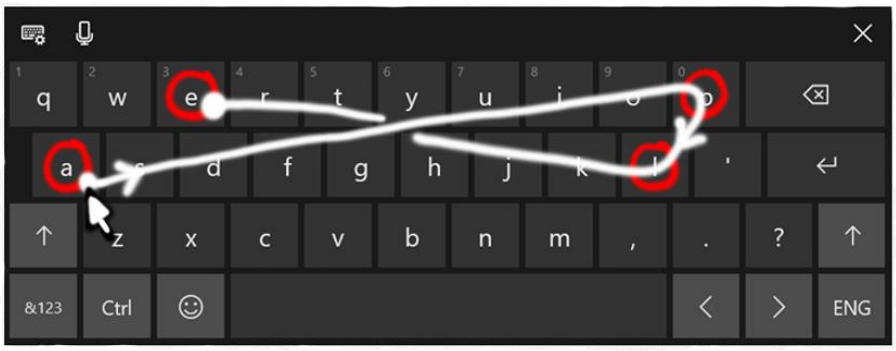
1. Press the oral button using your tongue to perform a left mouse click. Press and hold to drag.

7.2.4 Pressing the macro buttons

1. *(Initial setup only)* Customise the macros to your liking. The default macro actions are undo (Control + Z), calibrate gyroscope, and delete word (Control + Backspace) from left to right. Users are not recommended to remove the gyroscope calibration function from the macros since it is necessary to calibrate the gyroscope occasionally.
2. Using your chin, press the desired macro button to initiate the macro action.

7.2.5 Glide-typing

Users are encouraged to enable the glide-typing input keyboard on their computer to enhance speed and lower the precision skill floor. Note that some software does not support glide-typing.



1. Aim at the first letter of the desired word. For example, “apple”.
2. Press and hold the oral button.
3. Glide the cursor across each letter in the word consecutively in one motion without releasing the oral button.
4. When the last letter of the word is reached, release the oral button.
5. If you have made a mistake, you can choose a word from the autosuggestions above the keyboard to correct it.

Since glide-typing is equipped with predictive text, it is unnecessary to position the cursor to the exact position of each letter. For example, accidentally swiping the cursor to “R” instead of “E” is still tolerable. As long as the cursor reaches each letter closely enough, the keyboard can identify the user’s intention and type the desired word even with user inaccuracies.

7.3 Troubleshooting

If you encounter problems using the “***Paralyse need to type word*** **™**” input system, please refer to the following checklist.

- Ensure all wire connections are secure.
- Ensure the oral button is dry and is wrapped in a watertight manner.
- Ensure the gyroscope is mounted correctly and securely on the bridge of your glasses.

8 Appendix

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

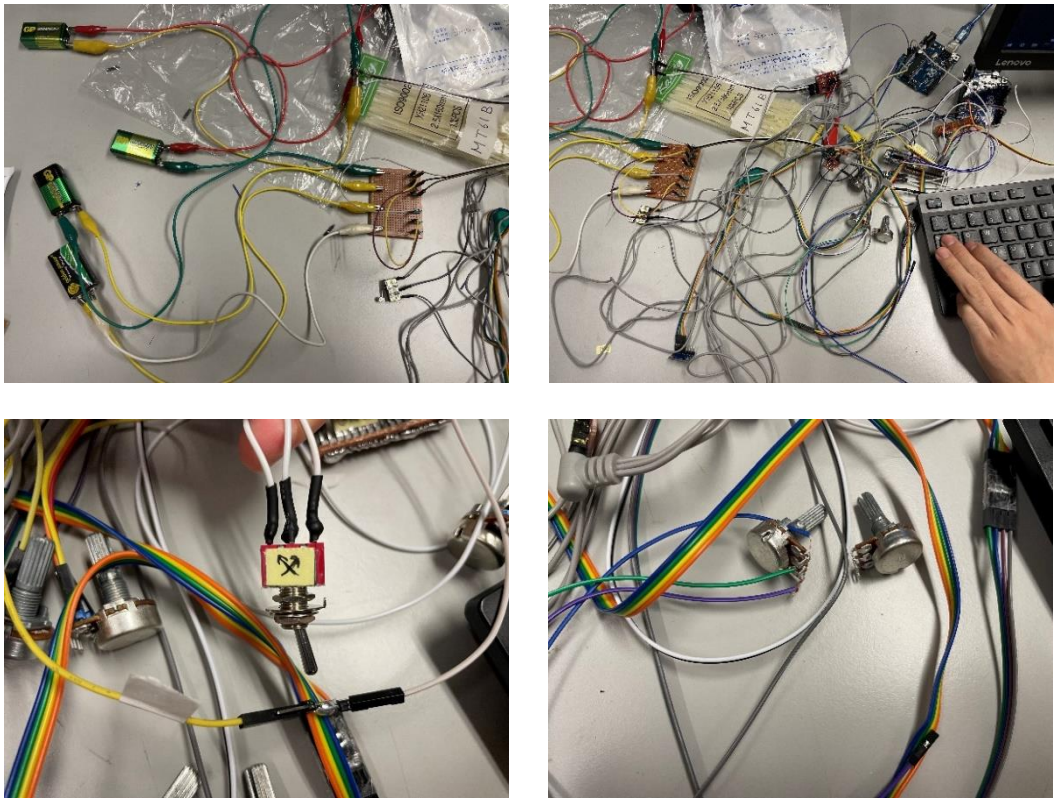


Figure 12
Circuits and wirings at different stages

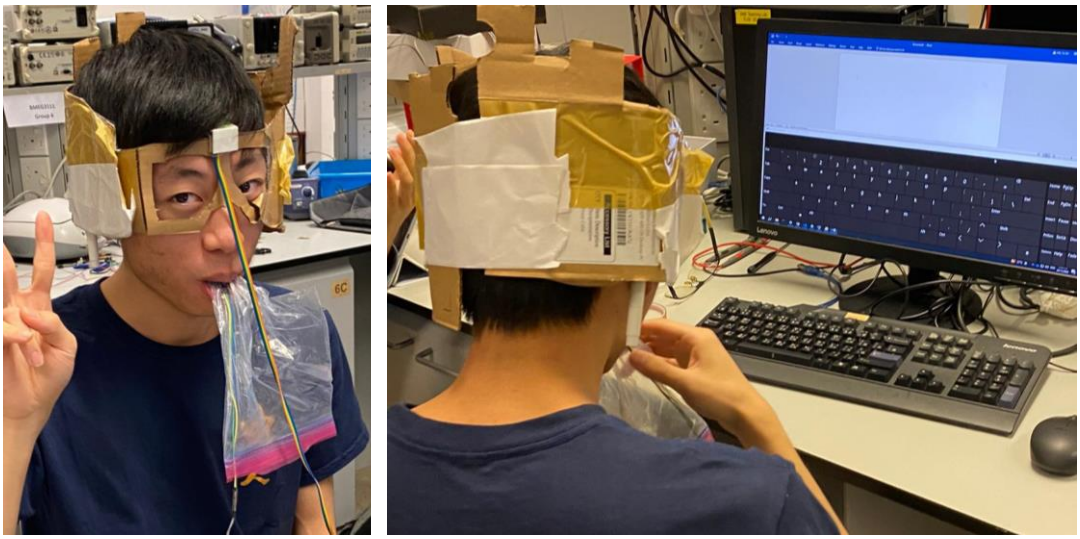


Figure 13
The first iteration of the prototype

9 References

[1] V. M. N. Passaro, A. Cuccovillo, L. Vaiani, M. De Carlo and C. E. Camanella, “Gyroscope Technology and Applications: A Review in the Industrial Perspective,” *Sensors,* vol. 17, no. 10, p. 2284, 7 October 2017.