**Software Development**

In designing an effective mouse control interface, we explored multiple approaches, including standalone **Python** scripts, external mapping software like **Karabiner-Elements**, and various Arduino libraries such as **AbsMouse, Mouse**, and **Keyboard**. These preliminary tests offered significant insights into managing inputs and simplifying control mechanisms. Given our objective to achieve seamless gameplay and responsive controls—particularly with four force sensors and an IMU—it was essential to prevent input lags and polling delays. Consequently, we opted for Arduino’s built-in Keyboard and Mouse libraries, enabling us to consolidate all controller commands within the microcontroller itself, thereby maximizing efficiency and responsiveness.

**Mouse Movement Control**

To provide a natural and intuitive control experience, a gyroscope mounted on the player’s head translates their head movements within their field of view (FoV). We used a complementary filter to refine the raw gyroscope and accelerometer signals, generating calibrated roll and pitch values suitable for mouse movement. Sensitivity settings were fine-tuned to ensure a balanced response, maintaining both stability and accuracy in horizontal and vertical directions.

**Thresholding and Double-Tap Logic**

* **Thresholding:** This method ensures movements are registered only when force or tilt inputs surpass specific thresholds, helping to minimize unintended actions.
* **Double-Tap Detection:** We developed logic to recognize double taps for certain commands by checking the timing between taps, ensuring it falls below a predefined limit (tapThreshold).
* **Debounce Logic:** A debounce mechanism was implemented to prevent unintentional rapid repeats, guaranteeing that each input is accurately recognized as a single intended action.

**Hardware Overview**

**Gyroscope (MPU6050)**

* **Logic:** The gyroscope’s roll and pitch data control the player’s in-game view, enabling natural camera movements. The code reads accelerometer and gyroscope data to calculate angles, which are then applied as camera adjustments.
* **Implementation:** The initializeMPU6050() function configures the sensor, while calibrateAccelerometer() and calibrateGyroscope() calibrate raw data by averaging multiple readings. The gyroSignals() function reads data, calculates angles, and applies a complementary filter to stabilize them. updateDeltaTime() calculates time differences between frames to ensure smooth integration of motion.

**Force Sensor Array**

* **Logic:** Each force sensor (F1, F2, F3, and F4) is linked to a specific movement key. For example, F1 triggers forward movement, while F3 initiates a leftward movement. Combining sensors allows for diagonal or more complex actions, such as jumping or flying, depending on pressure patterns.
* **Implementation:** The handleSensorInputs() function maps each sensor (F1 to F4) to corresponding actions based on thresholded readings. For instance, if F1 exceeds its threshold, the program interprets this as a “move forward” command by pressing the ‘W’ key. keyboardSingle() triggers individual keypresses, while keyboardCombo() allows two keys to be pressed simultaneously for diagonal or compound actions, like moving forward-right (W and D keys).
  + **Special Commands:** Through specific combinations, additional commands are created:
    - **Jump:** Triggered by pressing F1 and F2 together.
    - **Break:** Triggered by pressing F3 and F4 together.
    - **Fly:** Executed with a double-tap on F1 and F2, with continuous pressing for altitude control.

**Component Selection**

To ensure reliable data handling and control, we chose the following components:

* **MPU6050:** Selected for its combined accelerometer and gyroscope functions, which are essential for tracking head movements.
* **Resistive Force Sensors:** Four sensors provide gameplay control through foot pressure, enabling multiple interactions by mapping specific game commands to each sensor.
* **Arduino R4 Minima:** This ARM M4-based microcontroller provides sufficient processing power and features a built-in USB controller, making it ideal for seamless game integration.

**Calibration Process**

**Accelerometer Calibration**

* **Logic:** To improve the accuracy of head-tracking movements, we calibrated the accelerometer by averaging multiple readings to reduce sensor biases.
* **Implementation:** In calibrateAccelerometer(), the accelerometer reads data 2000 times, averaging values along each axis. This creates offsets for accCalibrationX, accCalibrationY, and accCalibrationZ, ensuring that future readings accurately reflect movements without bias.

**Gyroscope Calibration**

* **Logic:** Gyroscope calibration reduces constant drift, establishing a stable, real-time tracking of head movements.
* **Implementation:** The calibrateGyroscope() function calculates average RateRoll and RatePitch values over a designated number of samples, setting a baseline for accurate motion calculations.