Five Finger Game Punch

Pana Maria, 335CA

Introduction

The project consists of a glove that allows video game control through hand movements (for player movement) and the pressing of a button (for game interaction). Data is transmitted wirelessly to the computer via Bluetooth. The goal is to create an alternative method of interacting with games, based on motion and gestures, that is intuitive, portable, and easy to use without traditional controllers.

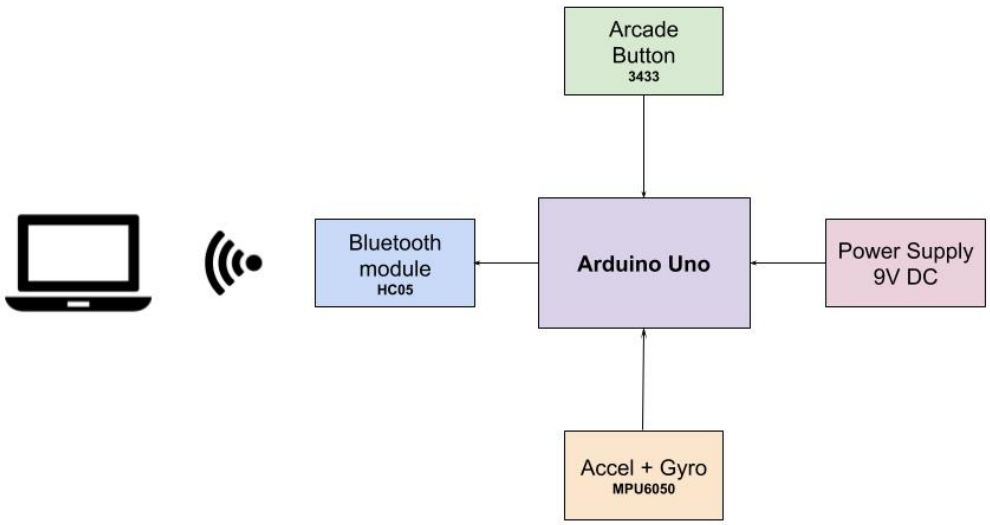
I wanted to explore more “natural” alternatives for interacting with devices, and video games - in this case, Minecraft - seemed like the ideal scenario for this. The usefulness of the project lies in its potential to become a hands-free alternative control solution, especially for mini-games, VR, or educational prototypes.

Description

The glove is equipped with an MPU6050 motion sensor (accelerometer + gyroscope), which detects the orientation and movement of the hand. These movements will be mapped to keyboard inputs as follows:

- Tilting the hand forward → **W** key
- Tilting the hand left → **A** key
- Tilting the hand backward → **S** key
- Tilting the hand right → **D** key

In addition, a button is placed near the thumb, allowing it to be easily pressed like a trigger (similar to a detonator). This button is mapped to an action key, such as left-click or attack, enabling interaction with the game environment (e.g., breaking blocks, hitting mobs).



The MPU6050 communicates with the Arduino Uno via the I2C protocol. The Arduino processes the data and transmits it via UART to an HC05 Bluetooth module. On the PC side, a program receives the Bluetooth data and emulates keyboard input, making the glove behave like a motion-enabled controller.

Hardware Design

Bill of Materials

Component	Description	Link
Arduino Uno	Microcontroller	Link
MPU6050	Accelerometer & Gyroscope Sensor	Link
HC05	Bluetooth Module	Link
3433	Arcade Button	Link
9V Battery	Power Supply	Link

Schematic

Software Design

Descrierea codului aplicaţiei (firmware):

- mediu de dezvoltare (if any) (e.g. AVR Studio, CodeVisionAVR)

- librării şi surse 3rd-party (e.g. Procyon AVRlib)
- algoritmi şi structuri pe care plănuiţi să le implementaţi
- (etapa 3) surse şi funcţii implementate

Used Pins

The MPU6050 gyroscope and accelerometer module is interfaced using the Arduino Uno’s I2C communication protocol. Specifically, pins A4 and A5 are used, where A4 (PC4) serves as the SDA (data line) and A5 (PC5) serves as the SCL (clock line). These pins are chosen because the ATmega328P microcontroller provides dedicated hardware support for I2C on these lines. The MPU6050 is powered via the 3.3V line and protected by a voltage divider.

For user interaction, an Arcade Button (3343) is connected with a combined LED and data line setup. The button's input is read on digital pin D8 (PB0), while the LED is powered from the 5V supply and grounded through GND. Pin D8 is strategically selected to avoid conflicts with critical communication pins - such as A4/A5 used for I2C, D0/D1 used for UART, and D10–D13 reserved for SPI - leaving these interfaces free for future expansion. Furthermore, D8 is compatible with Timer2-based interrupts, which are utilized for polling the button at 2kHz (every 0.5 milliseconds).

The HC05 Bluetooth module is interfaced via the Arduino’s hardware UART on pins D0 and D1. Specifically, the HC05’s RX pin is connected to Arduino TX (D1/PD1), and its TX pin to Arduino RX (D0/PD0). These are the default UART pins on the ATmega328P, providing direct access to the USART0 peripheral. The HC05 is powered via the 5V line and protected by a voltage divider.

Software Design

Although I initially started developing the code in Arduino IDE and used its provided libraries, once I got the project working I replaced the library code with my own implementations, targeting my use case and applying the knowledge gained during the laboratories.

At its core, the software is composed of two parts: the C code for the Arduino and the Python code for the PC.

Arduino Side

1. Initialization

Upon powering on, the Arduino initializes its hardware components with low-level control:

- **I2C Interface**

- Configured using *direct register manipulation* instead of the standard `Wire` library.
- Sends initialization commands to the **MPU6050** sensor to wake it from sleep and enable accelerometer and gyroscope output.

- **Bluetooth Communication (UART)**
  - Implemented using *bit-banged UART* on digital I/O pins.
  - Baud rate is controlled precisely using **timers** for accurate bit timing.
  - Originally used `SoftwareSerial`, but replaced due to interrupt conflicts.

2. Main Loop

The main loop handles continuous data processing and user input detection:

- **Sensor Data Handling**
  - Reads accelerometer values from the **MPU6050** over I2C.
  - Processes raw X and Y axis data.
  - Converts values into roll and pitch.

- **Command Generation**
  - Based on tilt direction, determines movement commands:
    - `W` = Forward
    - `A` = Left
    - `S` = Backward
    - `D` = Right
  - Ensures **mutually exclusive** movement — only one direction is sent at a time.

3. Button Handling

- **Button Input**
  - Connected to a digital pin with **interrupt-driven debounce** logic.
  - **Timer2** used to trigger debounce checks every 1 ms.
  - Differentiates between:
    - **Short press** = Left-click
    - **Long press** = Interpreted as multiple short presses

- **Timer2 Justification**
  - Initially used `SoftwareSerial`, which **disables all interrupts** during communication.
  - Timer2 was adopted as a workaround and retained even after switching away from `SoftwareSerial`.

4. Heartbeat Mechanism

- Sends a **heartbeat signal** (H) every 2 seconds.
- Ensures the PC side knows the Arduino is still connected and responsive.

PC Side (Python Script)

1. Serial Communication

- Opens a **serial connection** to the Bluetooth module (recognized as a virtual COM port).
- Continuously **listens** for incoming characters from the Arduino.

2. Command Handling

- **Movement Characters** (`W`, `A`, `S`, `D`):
  - Uses the `pynput` library to simulate key presses.
  - Smoothly switches between keys — previous key is released before a new one is pressed.
- **Mouse Click (`B`)**
  - Triggers a quick **left-click** event.
- **Heartbeat (`H`)**
  - Acknowledged by the script to confirm that communication is active.

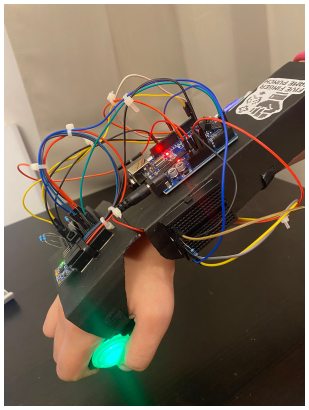
3. Performance Considerations

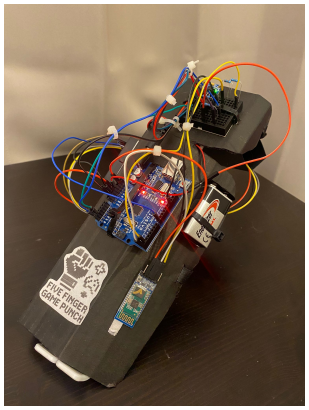
- **Non-blocking architecture**
- Python script runs a continuous loop with **event logging** for debugging and monitoring.

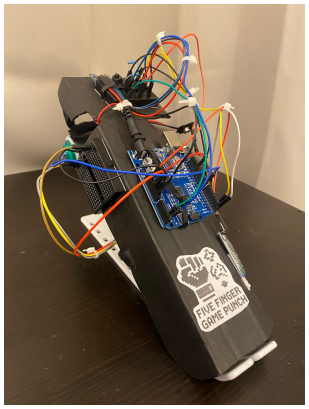
Conclusions & Demo

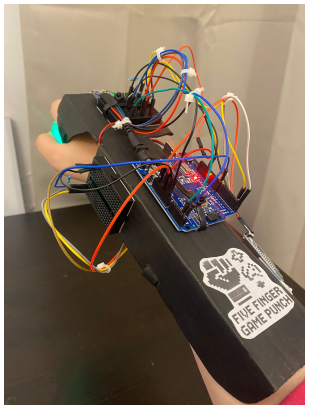
Now that the glove itself is also done, the “final product” is ready to go. The project works as intended i.e. both movement and interaction are controlled accordingly by the glove. To get a glimpse of how using it looks like, check out this demo:

Click here for demo video









Downloads

five_finger_game_punch_downloads.zip

Bibliography

Hardware Resources:

- ATmega328P Datasheet
- MPU6050 Datasheet
- HC05 Datasheet

Software Resources:

- Pynput
- Pyserial
- Computing angles for MPU6050

Miscellaneous resources (mainly for working out issues):

- Wiring Arcade Buttons
- Realising the only available solution to use the arcade button was to solder jumper wires to the button pins
- A helpful resource in trying to find out why the 9V battery wasn't powering the board (Hint: your board works fine, try a fresh battery)

Export to PDF