ECE 445

SENIOR DESIGN LABORATORY

DESIGN DOCUMENT

A Smart Glove for HCI

Team #34

HONGWEI DONG (hd2@illinois.edu)

SHANBIN SUN (shanbin3@illinois.edu)

JINHAO ZHANG (jinhaoz2@illinois.edu)

ZHAN SHI (zhans6@illinois.edu)

TA: Yu Yue

April 13, 2025

Contents

1 Introduction

1.1 Problem

In today's society, with the popularity of electronic devices such as laptops and smartphones, people's demand for efficient and convenient human-computer interaction is increasing[1]. However. However, traditional human-computer interaction methods (such as keyboards, mice, and touch screens) are often inefficient or limited in usage scenarios under some condition, and it is difficult to meet the current needs of multiple devices and multiple environments. For example, with the development of emerging technologies such as AR/VR, people require more natural and flexible gesture- and motion-based interactions[2]. But voice recognition and camera-based gesture recognition are limited by factors like noise, lighting, and distance. Therefore, How to ensure the accuracy of operation while using flexible interaction methods anytime and anywhere has become an urgent problem to be solved.

1.2 Solution

As a new type of human-computer interaction device, smart gloves can capture and identify the curvature, movement trajectory and spatial orientation information of fingers in real time, map gestures to various quick operations, and provide users with a convenient and efficient interactive experience. With multiple built-in sensors, smart gloves can accurately record the movement of fingers, and map the recognized gestures to pre-defined functions, thereby realizing convenient control of smart devices such as laptops, smartphones, and even AR/VR systems. There are many application scenarios. While working, users no longer need to switch frequently between their computer and presentation device. Instead, they can operate PowerPoint or documents with simple gestures. During gaming time, gesture interaction can provide a more immersive and natural experience; for people with disabilities, smart gloves can also help replace traditional input devices and improve the operation method. In this way, the user's hand movement information is converted into machine-recognizable instructions, which greatly improves the operation efficiency, and expands the application scope of human-computer interaction.

Due to time, manpower and resource constraints, we decided to focus on the most basic function - using smart gloves as a "mouse replacement" for laptops. Specifically, the built-in sensors on the gloves collect information about finger gestures, and map this information to mouse operations such as cursor movement, clicks or scroll wheels. Although the scope of this project is relatively limited, we will fully consider the scalability of the system during design and implementation so that it can be further expanded to more application scenarios in the future.

1.3 Visual Aid

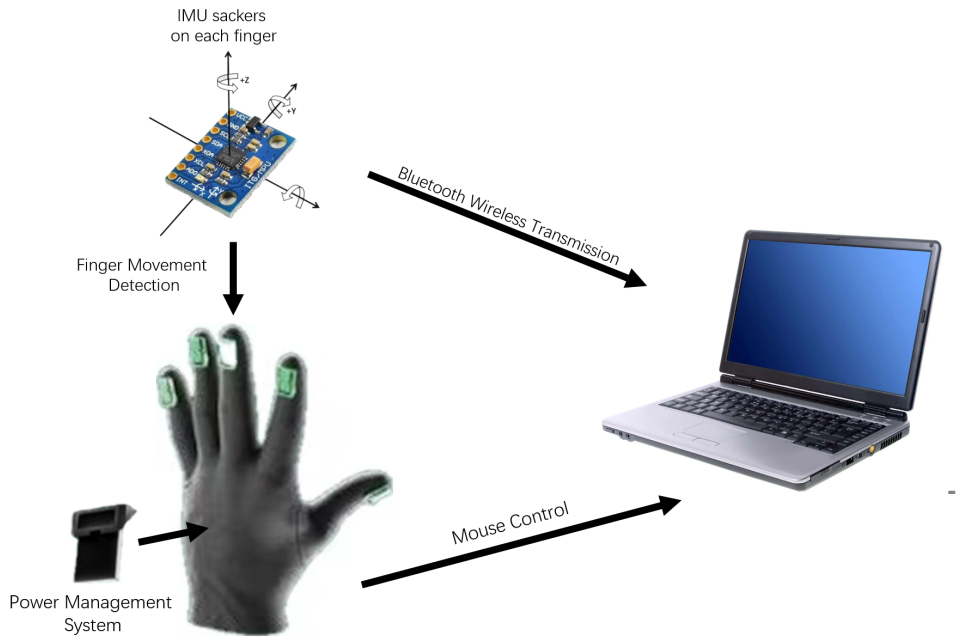

Figure 1: Visual Aid[3][4][5]

1.4 High-level Requirement List

1. The main function of this project is to use an Inertial Measurement Unit to map the position of the palm or index finger to the mouse pointer on the screen, enabling natural movement. At the same time, gestures such as clicking, pinching, and gripping are used as user commands, mapped to actions like left-click, right-click, and double-click, ensuring stability and reducing gesture misinterpretation or repeated commands.
2. The local gesture recognition system on the chip utilizes a pre-trained gesture recognition model to implement shortcut key mapping. To ensure real-time performance, our gesture recognition model must be fast and efficient, ensuring that each gesture command can be recognized and executed by the computer within 0.2 seconds.
3. As a wearable electronic device, we want this device to be portable and lightweight. Therefore, the proposed device weight will not exceed 0.5 kilograms to ensure that users can easily carry it during the HCI experience. To guarantee a long wireless usage experience for users, the glove should be able to operate for over 5 hours under normal usage conditions.

2 Design

2.1 Block Diagram

There are 4 components in our system, lithium battery and voltage regulator circuit, MPU6050 sensors, ESP32 MCU and the host device. The block diagram of the entire system is shown as follows 2.

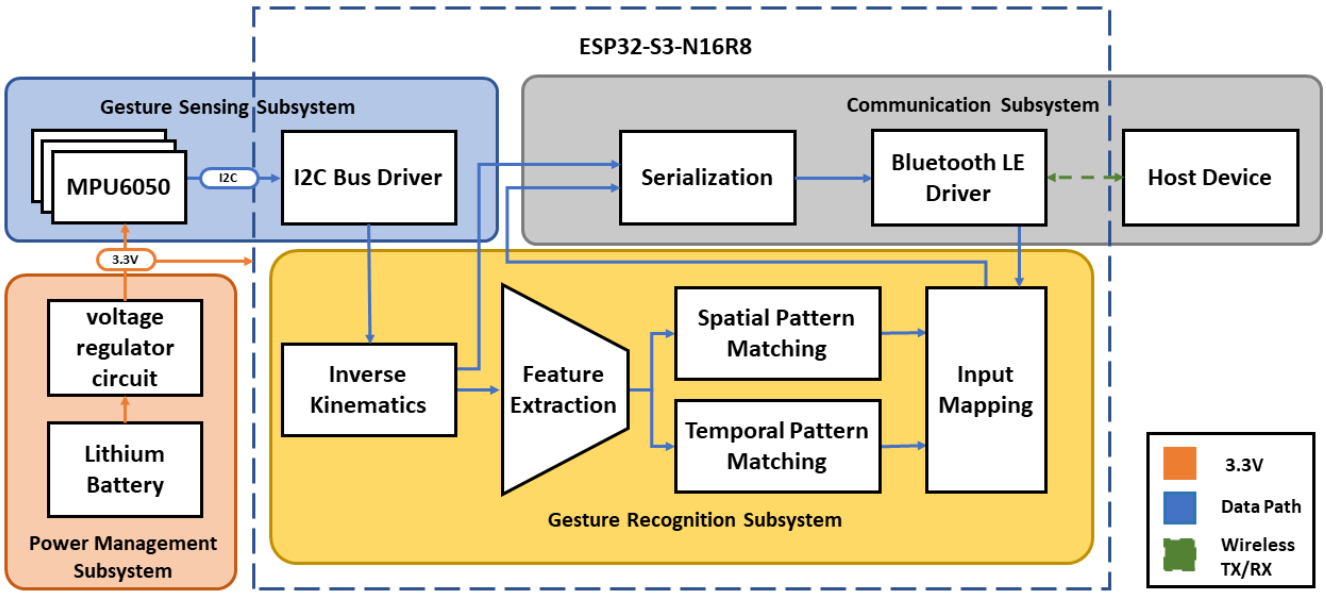

Figure 2: Block Diagram of System Design

2.2 Subsystem Overview

The smart glove is composed of four distinct subsystems: an IMU-based gesture sensing system, a gesture recognition system, a communication system, and a power management system.

2.2.1 IMU Based Gesture Sensing System

The gesture sensing system is responsible for capturing the position and the angle information of the fingertip. Each glove is equipped with six MPU6050 sensors, constituting a 6 DOF IMU. The communication infrastructure between the ESP32 and the MPU6050 is facilitated by an I2C bus. The I2C bus driver plays a pivotal role in regulating the SCL and SDA wires and implementing a built-in integral and Kalman filtering algorithm.

2.2.2 Gesture Recognition System

The goal of this system is to implement mouse movement based on the ESP32 and MPU6050, and map gestures (such as click, swipe, fist, etc.) to mouse buttons or keyboard shortcuts to accurately generate human-computer interaction commands. For the mouse movement part, we use a physical modeling approach, applying a series of coordinate transformations and integrations to the sensor data to obtain velocity, which guides the mouse movement. For the shortcut mapping, we adopt dynamic gesture detection rather than static gesture detection. We use preprocessed 3D acceleration and angular velocity data sequences from the MPU6050, which provides data for five fingers and the back of the hand, to construct a dynamic gesture database. During the user interaction phase, we sample sensor data at an appropriate frequency and use a sliding time window to obtain candidate sequences. We match this data sequence with the database using the DTW algorithm, and the matched gesture is mapped to a keyboard shortcut via HID.

2.2.3 Communication System

The communication system is designed to achieve efficient data interaction between smart gloves and computers or mobile devices. The system uses serialization and deserialization technology to encapsulate and parse data in binary or JSON format on the device and host sides to improve transmission efficiency. At the hardware level, the system integrates the CP2102/CH340 USB module, which supports wired communication of the UART interface, suitable for glove charging or scenarios requiring high-bandwidth data transmission; at the same time, the Bluetooth module provides wireless communication capabilities and supports Bluetooth (optional low-power Bluetooth LE) for the transmission of gestures and commands to enhance portability and user experience.

2.2.4 Power Management System

The power management system is designed to provide a stable power supply to the whole system. It is mainly divided into two parts: the battery charging section and the processor power supply section. The battery charging section will receive input voltage from USB/Type C interface and charging the battery under monitoring. The processor power supply section mainly includes the converter circuit to provide a stable 3.3v output voltage to the processor(Gesture Sensing Subsystem and Gesture Recognition Subsystem)

2.3 Hardware Design

2.3.1 Attitude Sensor PCB Design

The MPU6050 is an advanced six-axis MEMS motion tracking chip that integrates a three-axis gyroscope (programmable full-scale ±250°/s to ±2000°/s) and a three-axis accelerometer (configurable range ±2g to ±16g), and is equipped with a digital motion processor (DMP). The system-on-chip runs the sensor fusion algorithm in real time at a sampling

frequency of 1kHz, and synchronously outputs 16-bit raw inertial data and quaternion-based processed attitude vectors through an I2C digital interface (supporting standard 100kHz and fast 400kHz modes). Its core advantages include: the industry-leading 4×4 ×0.9mm QFN-24 package can be directly mounted on the knuckles; the ultra-low operating current of 3.6mA (5uA sleep mode) meets the battery life requirements of wearable devices; the built-in temperature compensation function ensures ±1% measurement stability in the range of -40°C to +85°C. These features fully meet the core design requirements of gesture recognition gloves: miniaturization, high energy efficiency, and stable tracking during dynamic hand movements.

In our IMU Attitude Sensor PCB design, MPU6050 is the core of the whole PCB. The VDD pin of MPU6050 requires a digital power supply ranging from 2.375V to 3.46V. It will be supplied with a 3.3V digital power supply from another power supply PCB. Pin VLOGIC is the optional logic level, typically connected to match the I2C bus voltage level (if different from the host controller's supply), so we connected it with VDD. SCL and SDA is two of the I2C communication port. SCL works as I2C clock input and SDA works as I2C data IO. Both of the pins are required to be connected with 4.7kΩ pull-up resistors. As we will not be using an external magnetometer, the pin XDA/XCL will not be connected. Pin AD0 is the I2C address selection port, so we default drop it down to the ground. Pin INT is the interrupt pin of MPU6050. It serves as a hardware-triggered event notification output, enabling real-time alerts to the host microcontroller (which, in our case, is ESP32) for specific sensor conditions without requiring continuous polling. It gives a high signal when the detected attitude changes. Because we will do continuous measurement, so we just leave pin INT floating. According to the datasheet, on pin 10 we used a regulator filter capacitor and on pin 13 we used a VDD bypass capacitor, both of which are valued 0.1μF. For Pin 20 a charge pump capacitor valued 2.2nF is needed. Finally, on VLOGIC pin 8, we need a bypass capacitor valued 10nF.

As the attitude sensor will be equipped onto human fingertips, the PCB needs to be rather small. To minimize its size, we organized some of the electronic components on the back of the board. Consequently, the real size of the board will be 9.9mm×12mm×1.6mm, which is actually smaller than a fingernail, enhancing the portability of the peripheral.

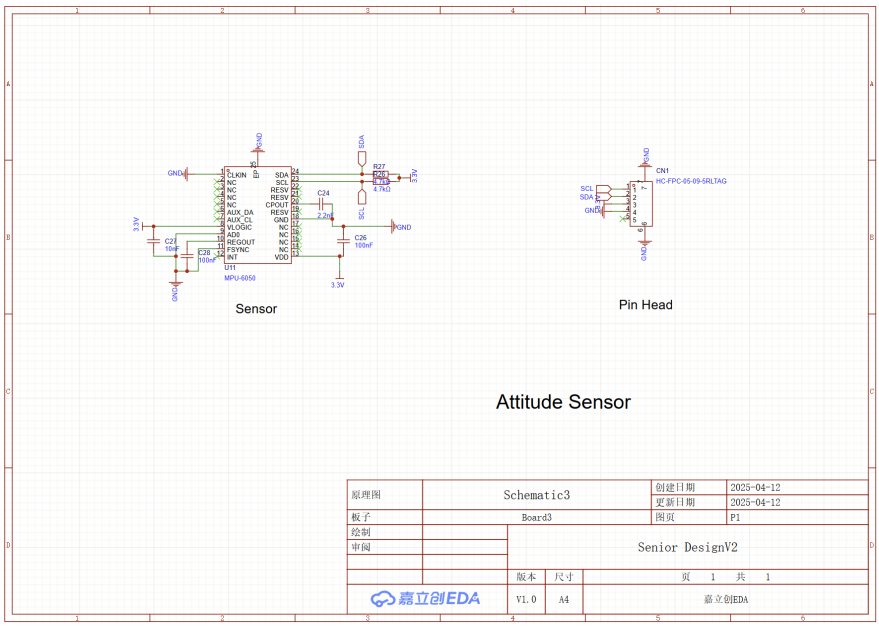

Figure 3: Schematic of Attitude Sensor

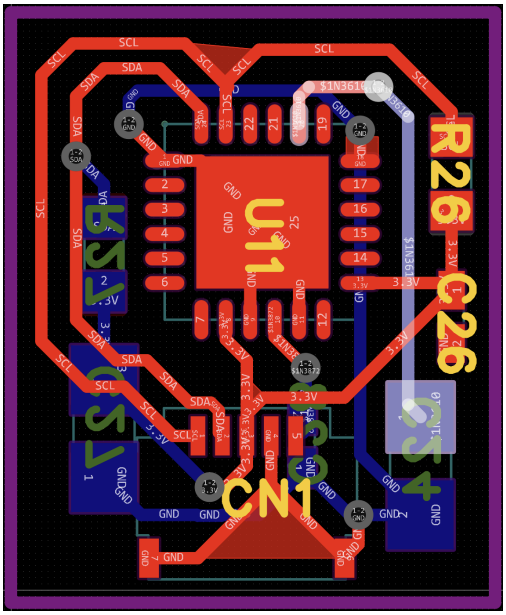

Figure 4: PCB Design of Attitude Sensor

2.3.2 I2C Switch PCB Design

We selected the **MPU6050** as the attitude sensor for our design. Since we need six MPU6050 modules (five on the fingertips and one on the palm) to achieve the attitude detection function, we must merge the six I2C signals into a single bus using a multiplexer.

To accomplish this, we adopted the TCA9548A chip. The TCA9548A is an I2C multiplexer primarily used to expand I2C bus connectivity and resolve address conflicts. It supports 1.8V, 3.3V, and 5V input logic levels, features low on-resistance (5Ω), and allows programmable addressing (default 0x70) via pins A0-A2. The chip is controlled via the I2C bus, can automatically isolate unselected channels to prevent bus conflicts, and is equipped with an interrupt output (INT) function to monitor downstream devices' status. It is particularly suitable for multi-sensor systems, modular designs, and mixed-voltage I2C network applications, simplifying bus expansion while ensuring signal integrity.

When the TCA9548A works as an I2C multiplexer, the host MCU sends a 1-byte channel selection command through the I2C bus, and the chip turns on the specified channel (the remaining channels remain in high-impedance state), achieving exclusive communication between the host and the target slave without causing bus conflicts. Its open-drain output INT pin can alert the host when a downstream device triggers an interrupt, thereby achieving efficient multi-device management without address conflicts.

The VCC pin of TCA9548A requires a voltage level ranging from 2.3V to 5.5V, so we will also use the same 3.3V supply from the voltage source as what we used in the design of attitude sensor. The SDA and SCL pins connect to the main I²C bus lines and require external pull-up resistors for proper operation. Three address configuration pins A0 through A2 determine the device's I2C slave address, with all pins grounded resulting in the default address of 0x70. An active-low RESET pin initializes the chip when pulled low, normally maintained at logic high during operation. The open-drain INT output pin signals interrupt conditions from downstream devices. Eight pairs of bidirectional SDx and SCx pins, where x ranges from 0 to 7, interface with the SDA and SCL lines of connected I²C slave devices respectively. Each channel can be individually selected through I²C commands while maintaining electrical isolation of inactive channels. The device's architecture supports hot insertion capability and requires properly sized pull-up resistors, typically 4.7kΩ, on the primary I2C bus lines to ensure reliable signal integrity across all connected devices.

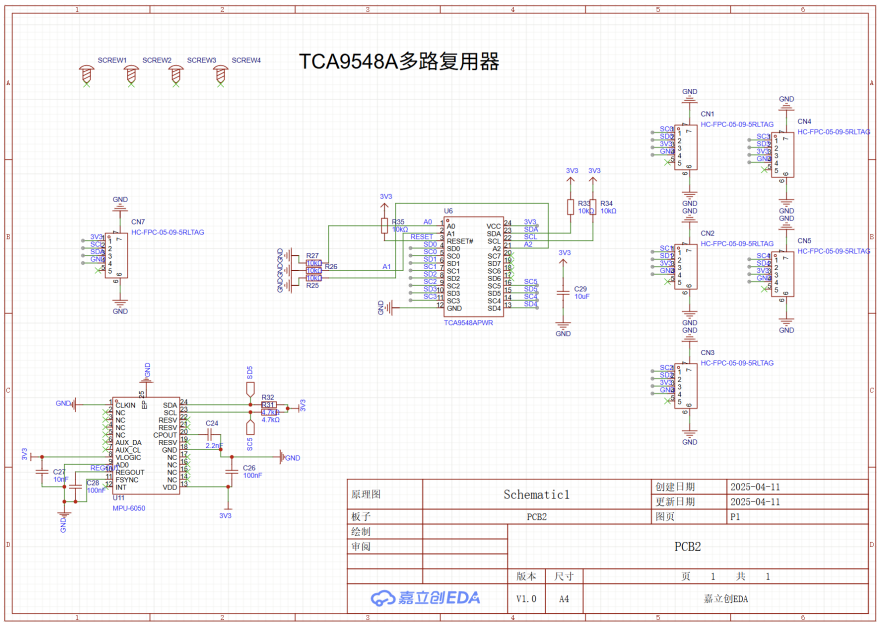

Figure 5: Schematic of I2C Multiplexer

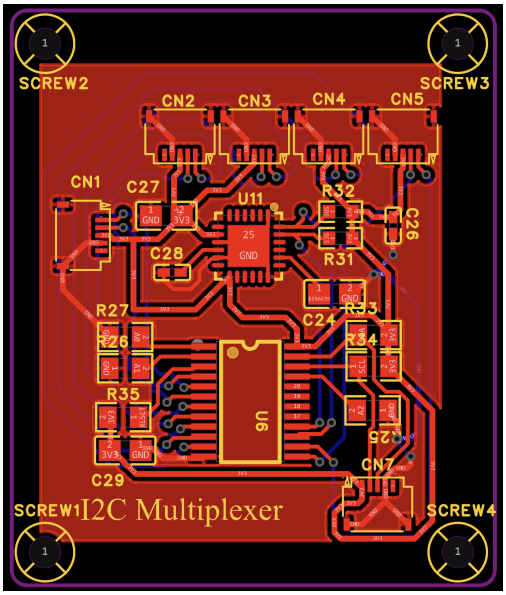

Figure 6: PCB Design of I2C Multiplexer

2.3.3 Power Management System

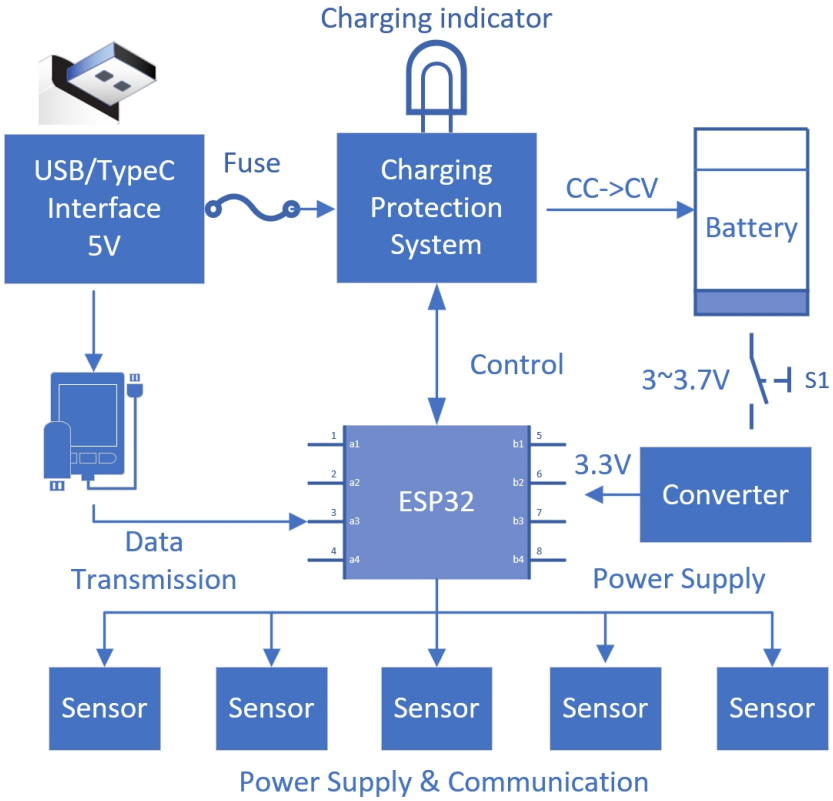

Figure 7: Power Management System Flowchart

As shown in Figure 7, the power management system is divided into two main sections: the battery charging section and the processor power supply section.

Battery Charging Section

- The system receives a 5V power supply via the Micro USB/Type-C interface, with a fuse added for overcurrent protection.
- The charging module uses a CC-CV mode charging IC from the LTH series to manage the battery charging process.
- Two charging indicator LEDs are included to display charging status.
- A charging protection chip is added to prevent overcharge and overdischarge, ensuring charging safety.
- The use of a Micro USB connector helps simplify later software integration and coding.

Processor Power Supply Section

- The converter system regulates the lithium battery's output voltage, to provide a regulated 3.3V $\pm$ 0.05V output to power the ESP32 and its peripherals.
- The ESP32 shall supply power to the connected sensors and other peripherals.

The following section provides a detailed description of the power supply modules

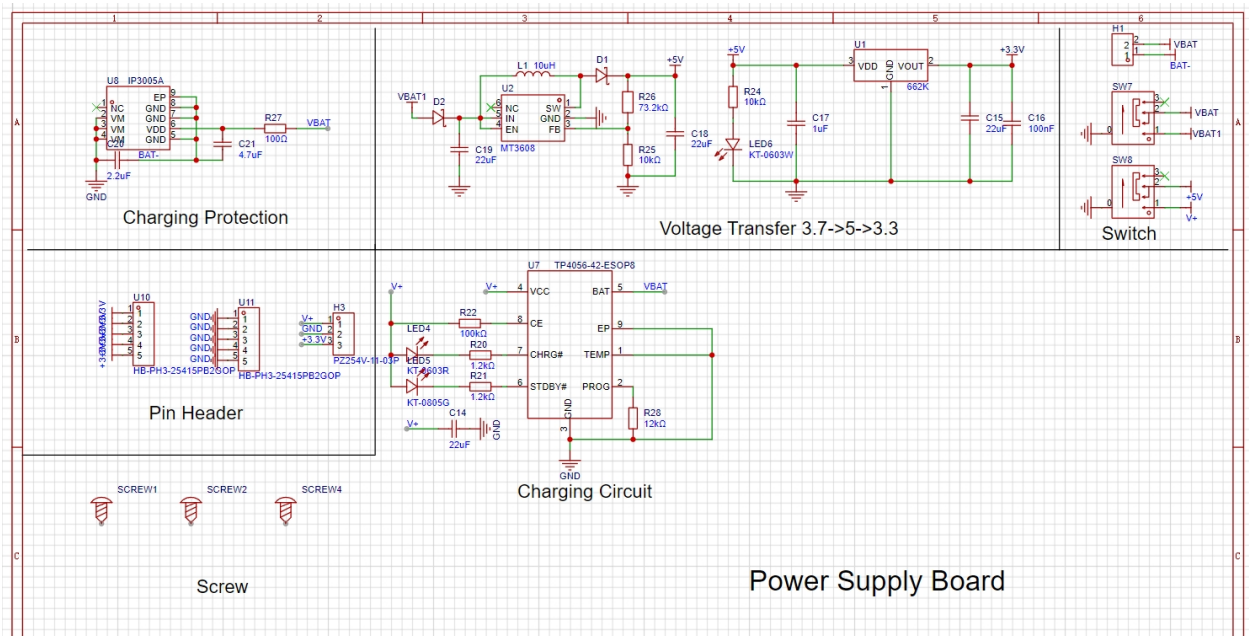

Figure 8: Power board schematic

The power supply board consists of six functional sections:*Lithium-ion battery*, *Charging Protection*, *Charging Circuit*, *Voltage Transfer*, *Pin Header*, and *Switch*. The detailed schematic graph is showcased in Figure8,The PCB board 2D and 3D graph can be seen in Figure9 and Figure17.

- Lithium-ion Battery: The selected battery is a 300mAh lithium-ion cell with dimensions of 25 mm $\times$ 30 mm $\times$ 4 mm, chosen to balance capacity and compact size. Its nominal charge/discharge rate is 0.2C, with a safe operating rate of 0.5C and a maximum rate of 1C. The physical appearance of the battery is shown in figure11.
- **Charging Circuit:** The TP4056 (U7) is used to implement a constant-current/constant-voltage (CC-CV) charging scheme. The charging current is set by the resistor R28 according to the formula $I = \frac{1\,\mathrm{V}}{R_{28}} \times 1200$, yielding approximately 100 mA for $R_{28} = 12\,\mathrm{k}\Omega$, which is within the recommended 0.2C–0.5C range for the selected lithium battery. During charging, the red LED (LED4) connected to `CHRG#` is lit; once charging completes, the green LED (LED5) connected to `STDBY#` turns on.
- **Charging Protection:** The IP3003A (U8) provides battery protection features, including overcharge, over-discharge, and short-circuit protection. The battery is connected via `BAT+` and `BAT−`, and the protected output is provided through `VBAT`.
  - Overcharge protection: 4.28 V; recovery: 4.1 V

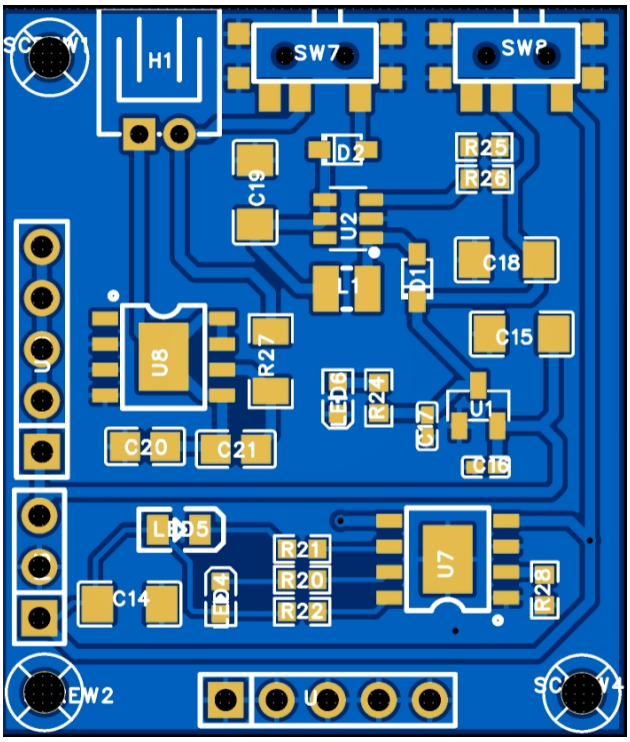

Figure 9: Power board PCB

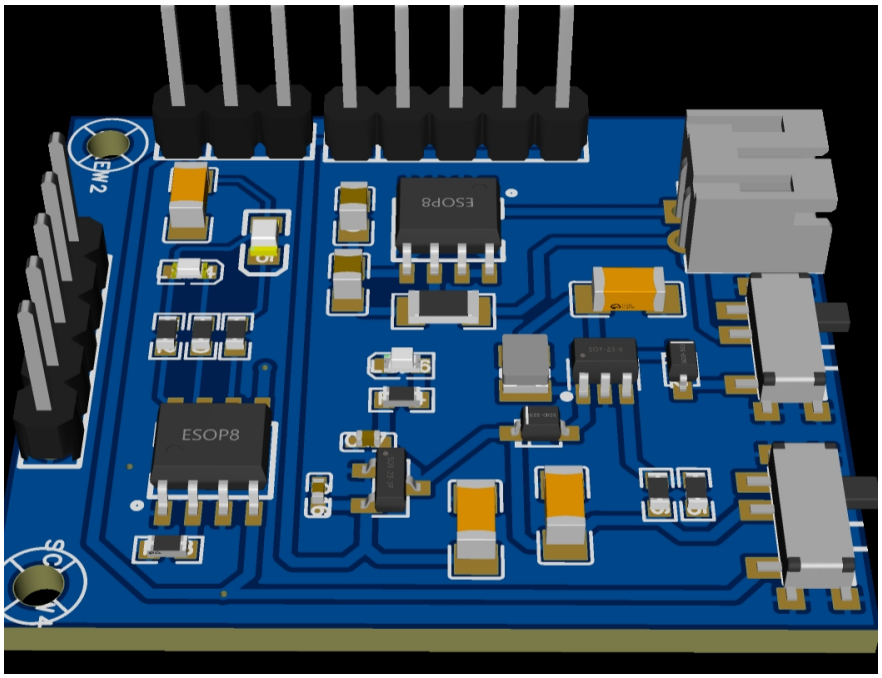

Figure 10: Power board PCB-3D View

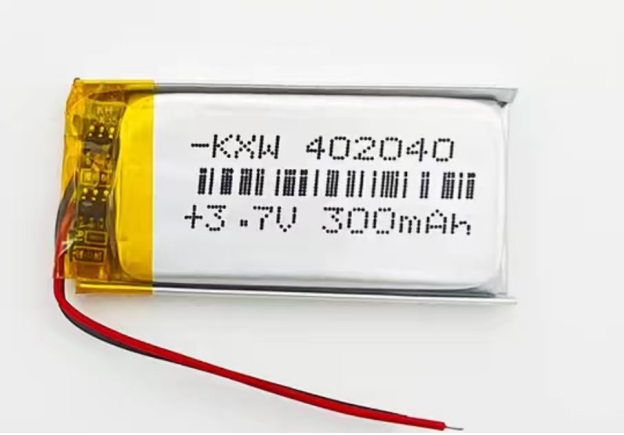

Figure 11: battery

– Over-discharge protection: 2.5 V; recovery: 3.0 V

These fixed thresholds align with lithium-ion battery specifications and effectively prevent unsafe voltage excursions, thereby improving battery longevity and safety.

- **Voltage Transfer (3.7 V $\rightarrow$ 5 V $\rightarrow$ 3.3 V):** To ensure a stable 3.3 V output, a two-stage voltage conversion scheme is adopted. First, the MT3608 boost converter (U2) raises the battery voltage (typically 3.7 V) to 5 V. The output voltage is set using the feedback formula:

$$V_{\text{OUT}} = 0.6\,\text{V} \times \left(1 + \frac{R_1}{R_2}\right)$$

With $R_1 = 73\,\text{k}\Omega$ and $R_2 = 10\,\text{k}\Omega$, the output is regulated near 5 V.

The second stage uses a 662K low-dropout linear regulator (U1) to step down 5 V to 3.3 V, with output smoothing by capacitors C15 and C16. This two-stage design ensures a reliable 3.3

- **Pin Header:** The board includes one 3-pin and two 5-pin headers for interfacing. The 3-pin header (H3) connects directly to the ESP32 board, providing 3.3 V and GND, and receiving 5 V input. This compact design supports easy stacking with the main controller board. The two 5-pin headers are reserved for future expansion, offering extra power and signal connectivity to external modules.
- **Switches:** Power routing is managed by two slide switches (SW7, SW8) and one terminal block (H1). SW7 controls whether the battery supplies power through the `VBAT` line. SW8 switches the 5 V USB supply, which is useful during firmware flashing or testing. The terminal block H1 is used to connect an external lithium-ion battery. This configuration allows flexible switching between USB-powered and battery-powered modes, enhancing both safety and development convenience.

2.3.4 ESP32 Board Design

The ESP32 Board consists of a Micro USB interface, which supplies 5 V input to the power management board and simultaneously receives a regulated 3.3 V output from it to power the onboard ESP32 module.

The development board integrates USB-to-serial circuitry and supports firmware downloading and serial communication. It also includes the necessary current path and logic for uploading code via USB. The schematic of the board is shown in Figure12, while the corresponding 2D and 3D PCB layouts are provided in Figure13 and Fiture14, respectively.

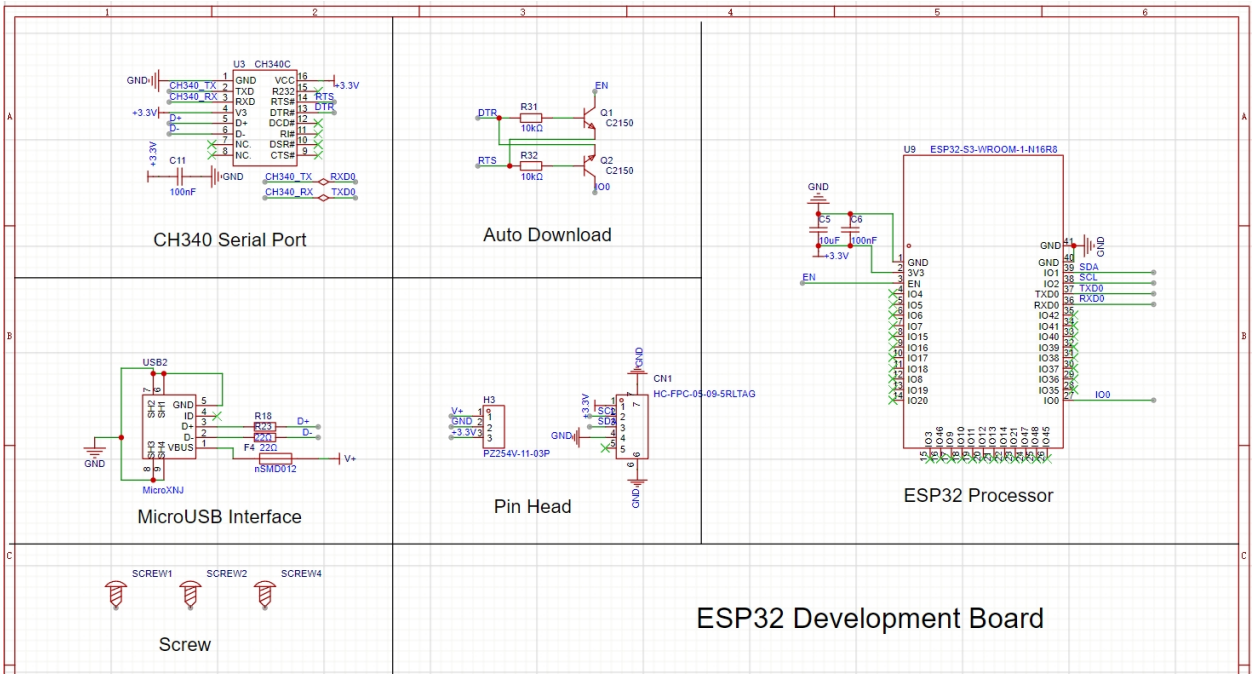

Figure 12: ESP32 Development Board

In the following section, we will provide a detailed explanation of its working principles and circuit design.

This schematic illustrates the design of the ESP32 core board, which integrates USB communication, serial port conversion, automatic download control, and the ESP32-S3 processor. The board consists of five functional blocks: *MicroUSB Interface*, *CH340 Serial Port*, *Auto Download Circuit*, *Pin Header*, and *ESP32 Processor Module*.

– **MicroUSB Interface:** The USB2 connector provides both power supply and data communication for the system. To protect the downstream battery and circuits, a fuse (F4) is added to limit the current below 290 mA. If the current exceeds this threshold, the fuse will blow, effectively preventing potential damage caused by overcurrent conditions

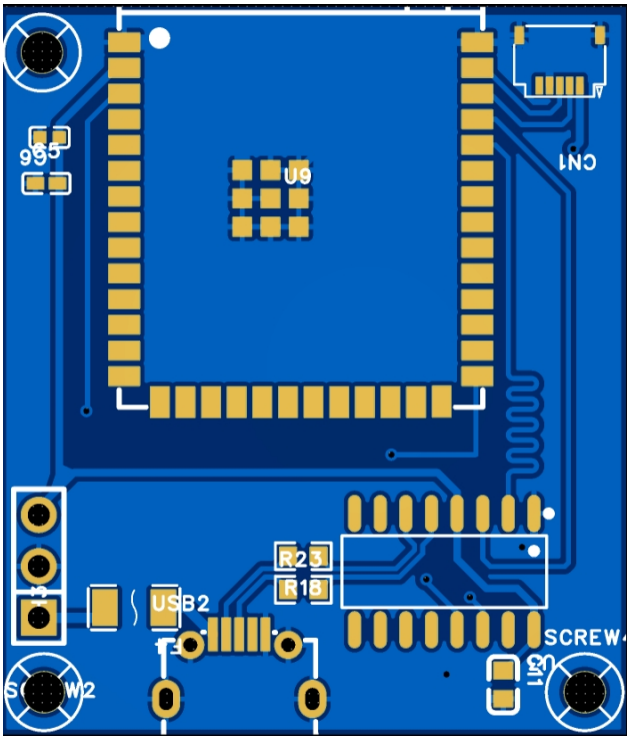

Figure 13: ESP32 Development Board PCB

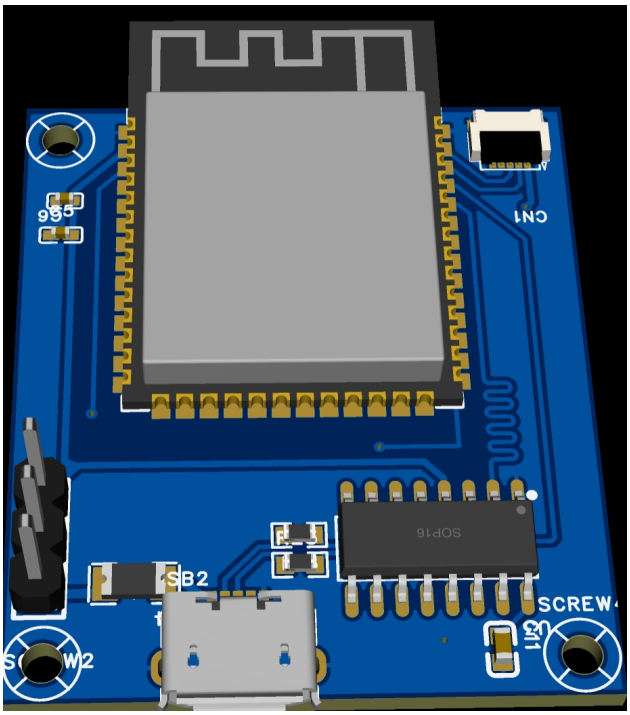

Figure 14: ESP32 Development Board PCB-3D View

- **CH340 Serial Port:** A CH340C chip (U3) is used to convert USB to UART, enabling serial communication between the host computer and the ESP32 module. It connects USB signals `D+` and `D−` to UART signals `TXD0` and `RXD0`.
- **Auto Download Circuit:** The automatic download circuit consists of two NPN transistors and pull-up resistors . This circuit uses the `DTR` and `RTS` signals from the USB-UART converter to automatically toggle the `EN` (reset) and `IO0` (boot) pins of the ESP32 for flashing firmware without manual intervention.
- **Pin Header:** The 3-pin header (H3) is designed to mate with the power supply board and is responsible for delivering power (`V+`, `GND`, and `3.3V`) to this module. The 5-pin FPC connector (CN1) is used to route the output signals from the CH340 and ESP32 modules—specifically the I^2C lines (`SDA`, `SCL`) and power rails—to the TCA9848A I^2C expander, enabling communication with multiple downstream sensors.
- **ESP32 Processor:** The ESP32-S3-WROOM-1-N16R8 module (U9) serves as the main controller. It operates on 3.3 V and connects to the serial port via `TXD0` and `RXD0`, with the `EN` and `IO0` pins interfaced to the download circuit.

2.3.5 Communication System

1. **Bluetooth Section** We will be using Build-in Bluetooth of ESP32 called BluetoothSerial to transmit data between the mircoprocessor and our computer. ESP32 supports Bluetooth 5.0 which has both Bluetooth and BLE (Bluetooth Low Energy), and we will use BLE for our project. Compared with BR, BLE has lower connection delay and lower power consumption, so it's more suitable on low power devices such as wearable devices. Bluetooth 5.0 has a transmission rate up to 2 Mbps and transmission distance up to 300 meters. In practical, the effective transmission rate is usually lower than theoretical value, which is about 1 Mbps.
2. **UART Section** UART is a serial communication protocol that transmits data to a host computer through a USB interface such as type-c. We use a CH340 serial communication chip to realize the support of UART. Through the UART protocol, we can monitor and program the ESP32. The PCB design for CH340 and ESP32 is shown as 14.

2.4 Software Design

2.5 Overview

Our software design has three main parts, I2C and IMU driver, gesture recognition module and BLE driver. In order to maintain the high performance and reproducibility of our projects, we adhere to the following guidelines in the design of our software.

1. **Zero Runtime Overhead Principle**. Any template class is designed to maximize the benefits of C++'s template programming, such as compile-time derivation using

constexpr.

2. **Abstraction Without Sacrificing Performance**. Abstraction of any object should not come at the expense of performance. By using programming paradigms such as CRTP (Curious Recursive Template Pattern), RAII (Resource Acquisition Is Initialization), we can achieve zero-cost abstraction of class objects and hardware

3. **Standardization and Portability**. The Standard Template Library (STL) provides a rich set of reusable components (containers, algorithms, iterators) that promote portability and reduce the need for reinventing the wheel. By using STL, all of our modules are header-only, providing extremely strong portability.

The relationship between the I2C driver, IMU driver and bluetooth driver is shown as follows.

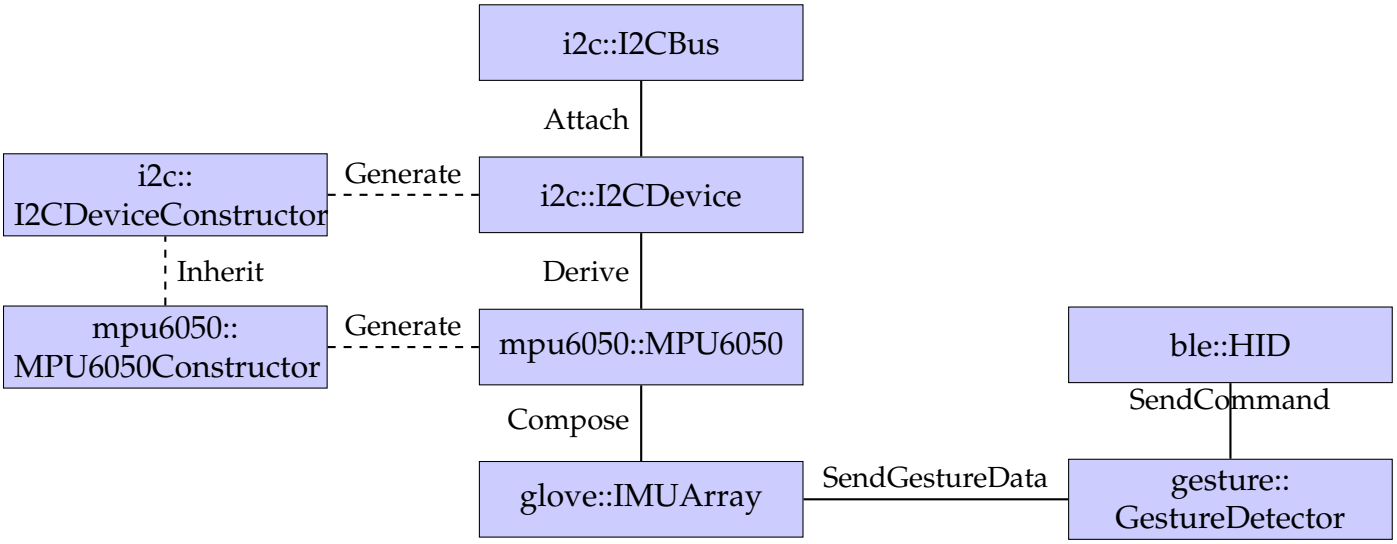

Figure 15: Datapath Flowchart

2.5.1 I2C Driver and IMU Driver

The i2c namespace provides a robust abstraction for managing I2C communication on an ESP32 device. It includes classes for configuring and interacting with I2C buses and devices:

1. I2CBus: Represents an I2C bus. Provides functionality to scan for connected devices and manage the bus lifecycle. Automatically cleans up resources when the bus is destroyed.

2. I2CBusConstructor: A builder class for creating and configuring an I2CBus instance. Allows customization of parameters like clock source, glitch ignore count, interrupt priority, and internal pull-up resistors.

3. I2CDevice: Represents an I2C device connected to a bus. Provides methods for reading and writing data to the device, including support for combined read-after-write

operations. Ensures proper cleanup of the device handle upon destruction.

4. I2CDeviceConstructor: A builder class for creating and configuring an I2CDevice instance. Allows customization of parameters like clock speed, wait time, and acknowledgment checks. The driver uses ESP-IDF's I2C APIs and ensures error handling . It supports both 7-bit and 10-bit addressing modes and provides a clean, modern C++ interface for I2C communication.

The mpu6050 namespace provides a high-level interface for interacting with the MPU6050 IMU (Inertial Measurement Unit) sensor. It leverages the i2c driver for communication and includes the following components:

1. Register Abstraction: The DEFINE_REGISTER macro simplifies the definition of sensor registers with bitfield structures and default values. The Reg template class provides a type-safe way to read and write specific registers.
2. MPU6050 Class: Encapsulates the functionality of the MPU6050 sensor. Provides methods to: Read gyroscope, accelerometer, and temperature data. Configure sample rate, full-scale ranges, sleep mode, and low-pass filters. Enable low-power mode with customizable configurations. Reset the sensor to its default state. Internally manages sensitivity scaling based on the configured full-scale ranges.
3. MPU6050Constructor: A builder class for creating and configuring an MPU6050 instance. Allows chaining of configuration methods for parameters like sample rate, full-scale ranges, sleep mode, and low-power mode. Automatically initializes the sensor with the specified settings upon construction.
4. Enums and Configurations: Enumerations for gyroscope and accelerometer full-scale ranges, low-pass filter configurations, and low-power wake frequencies.

The driver provides a clean and efficient interface for working with the MPU6050 sensor, making it easy to configure and retrieve data while abstracting low-level I2C communication details.

2.5.2 Bluetooth Low Energy Driver

The Bluetooth driver is abstracted using the ESP-IDF's BLE (Bluetooth Low Energy) stack, specifically tailored for HID (Human Interface Device) profiles. Here's how the abstraction works:

1. HID Profile Initialization:

   The esp_hidd_profile_init() function initializes the HID profile environment (hidd_le_env) and sets up the necessary BLE services and characteristics for HID functionality. The HID profile includes predefined characteristics for mouse, keyboard, and consumer control input reports.

2. Callback Registration:

   The esp_hidd_register_callbacks() function allows the application to register event

callbacks for handling HID-specific events (e.g. connection, disconnection, report writes). These callbacks are stored in the hidd_le_env structure and invoked during BLE events.

3. Report Handling: The hid_dev_register_reports() function registers HID reports (e.g., mouse, keyboard, consumer control) with their corresponding IDs, types, and handles. The hid_dev_send_report() function sends HID reports to the connected BLE client using the appropriate GATT handle
4. BLE GATT Services: The HID service and its characteristics are defined in hidd_le_gatt_db and include attributes for input reports, output reports, and control points. The hidd_le_create_service() function sets up the GATT database and starts the HID service.
5. Event Handling: The gatts_event_handler() function handles BLE GATT server events (e.g., registration, connection, disconnection, attribute writes) and routes them to the appropriate handlers. Setting Up an HID Device to Simulate Input

To simulate an HID input device (e.g., a mouse or keyboard) using the BLE driver shown as above, the following steps are performed.

1. HID Profile Initialization:

   Call esp_hidd_profile_init() to initialize the HID profile. This sets up the BLE GATT services and characteristics required for HID functionality.

2. Register HID Reports:

   Use hid_dev_register_reports() to register the HID reports (e.g., mouse, keyboard) with their IDs, types, and GATT handles. For example, a mouse input report is registered with HID_RPT_ID_MOUSE_IN.

3. Send HID Reports:

   Use functions like esp_hidd_send_mouse_value() or esp_hidd_send_keyboard_value() to send HID input reports to the connected BLE client. For a mouse: esp_hidd_send_mouse_value(conn_id, mouse_button, mickeys_x, mickeys_y) sends a report with button states and movement deltas (mickeys_x, mickeys_y). For a keyboard: esp_hidd_send_keyboard_value(conn_id, special_key_mask, keyboard_cmd, num_key) sends a report with key presses.

4. BLE Connection Management: The HID device handles BLE connections and disconnections through the registered callbacks. For example, when a BLE client connects, the HID device starts sending input reports.

2.5.3 Gesture Recognition System

Keypoints:

1. Overall, our human-computer interaction is divided into two parts: basic mouse movement and shortcut keys. Mouse movement is achieved by deriving velocity

from sensor data through a series of coordinate transformations and integrations, while shortcut key mapping is accomplished through a designed model and the DTW algorithm.

2. In shortcut key mapping part, we predefine gestures as specific sensor time series data, and use the DTW algorithm to match the sensor data from each time window with the predefined database to recognize the gesture and map it to a mouse or keyboard shortcut.

3. Our primary goals are high accuracy and real-time performance. In terms of accuracy, we use a low-pass filter or a Kalman filter to process the raw sensor data to eliminate noise, and average the data of multiple users to eliminate individual differences. At the same time, real-time performance is achieved by quantizing the sensor data into integers and storing them in the NVS(Non-Volatile Storage), which is implemented based on the internal flash memory, reducing the data transmission time and latency.

Mouse movement For the mouse's X-axis and Y-axis movements, we utilized the accelerometer data from the MPU6050 mounted on the back of the hand. However, since the MPU6050 is affected by gravity and experiences a downward acceleration of g, we first use the gyroscope to compensate for the gravitational acceleration. The specific process is as follow:

The gravity vector in the world coordinate frame, the raw accelerometer and the gyroscope measurement are:

$$\boldsymbol{g}_{\text{world}} = \begin{bmatrix} 0 \\ 0 \\ -9.81 \end{bmatrix} \qquad \boldsymbol{a} = \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} \qquad \boldsymbol{\omega} = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}$$

The gyroscope provides the angular velocity components ω_x, ω_y, and ω_z, which represent the rates of change of the Euler angles ϕ, θ, and ψ, corresponding to roll, pitch, and yaw, respectively. The relationship between angular velocities and Euler angles is:

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & \sin\theta\tan\phi & \cos\theta\tan\phi \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta/\cos\phi & \cos\theta/\cos\phi \end{bmatrix} \cdot \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}$$

Once we have the angular velocities ω_x, ω_y, and ω_z, we can integrate them numerically to obtain the Euler angles $\phi(t)$, $\theta(t)$, and $\psi(t)$ at each time step:

$$\phi[k+1] = \phi[k] + \dot{\phi}[k] \cdot \Delta t$$
$$\theta[k+1] = \theta[k] + \dot{\theta}[k] \cdot \Delta t$$

$$\psi[k+1] = \psi[k] + \dot{\psi}[k] \cdot \Delta t$$

The rotation matrix $\boldsymbol{R}$ (from world frame to sensor frame) is constructed from the Euler angles:

$$\boldsymbol{R} = \begin{bmatrix} \cos\psi\cos\theta & \cos\psi\sin\theta\sin\phi - \sin\psi\cos\phi & \cos\psi\sin\theta\cos\phi + \sin\psi\sin\phi \\ \sin\psi\cos\theta & \sin\psi\sin\theta\sin\phi + \cos\psi\cos\phi & \sin\psi\sin\theta\cos\phi - \cos\psi\sin\phi \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{bmatrix}$$

The gravity vector in the sensor coordinate frame is obtained by:

$$\boldsymbol{g}_{\text{sensor}} = \boldsymbol{R}^{\top} \cdot \boldsymbol{g}_{\text{world}}$$

Therefore, the net acceleration after removing gravity is:

$$\boldsymbol{a}_{\text{no\_gravity}} = \boldsymbol{a} - \boldsymbol{g}_{\text{sensor}}$$

Therefore, we can integrate the gravity-compensated acceleration to obtain the velocity, and use the velocity in the x and y directions as the speed for mouse movement in the horizontal and vertical directions.

$$\boldsymbol{v}_{k+1} = \boldsymbol{v}_k + \boldsymbol{a}_{\text{no\_gravity},k} \cdot \Delta t$$

Data preprocessing and database construction.

Unlike simple mouse movement, gesture mapping is achieved through a dynamic mapping model. The data output by the MPU6050 may be subject to noise interference. By using the low-pass filter to the data from the accelerometer and gyroscope, the influence of noise can be effectively reduced, and the accuracy and stability of pose matching can be improved. Then, to facilitate faster inference and adapt to the NVS storage format, we quantize the original floating-point values into integer form. The accelerometer data quantization relationship is shown in the figure.

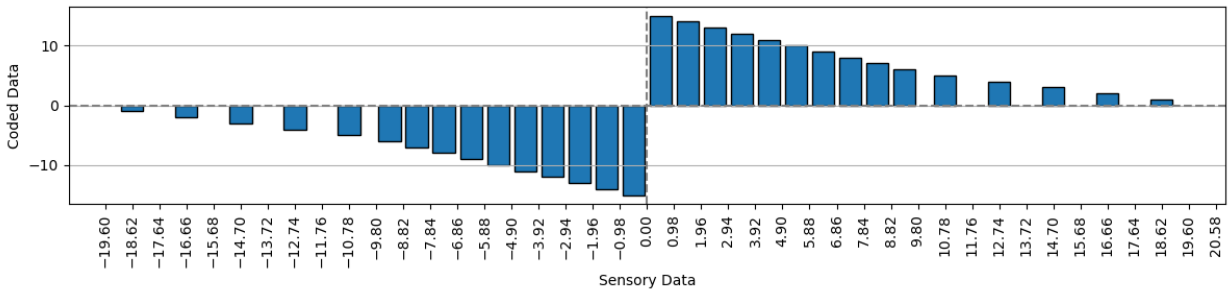

Figure 16: Quantization relationship

We also perform similar quantization on the gyroscope data. Then, we define several commonly used mouse and keyboard commands—such as clicking, increasing volume and so on—as gestures. For each gesture, we collect multiple samples from multiple users and use the average as the representative data sequence for that gesture. Therefore, the data corresponding to each gesture has a dimension of $[num\_sensors * 6, sequence\_length]$, where 6 means the dimension of the accelerometer and the gyroscope data. The entire database is stored in NVS(Non-Volatile Storage) in the form of key-value pairs.

The process of gesture recognition

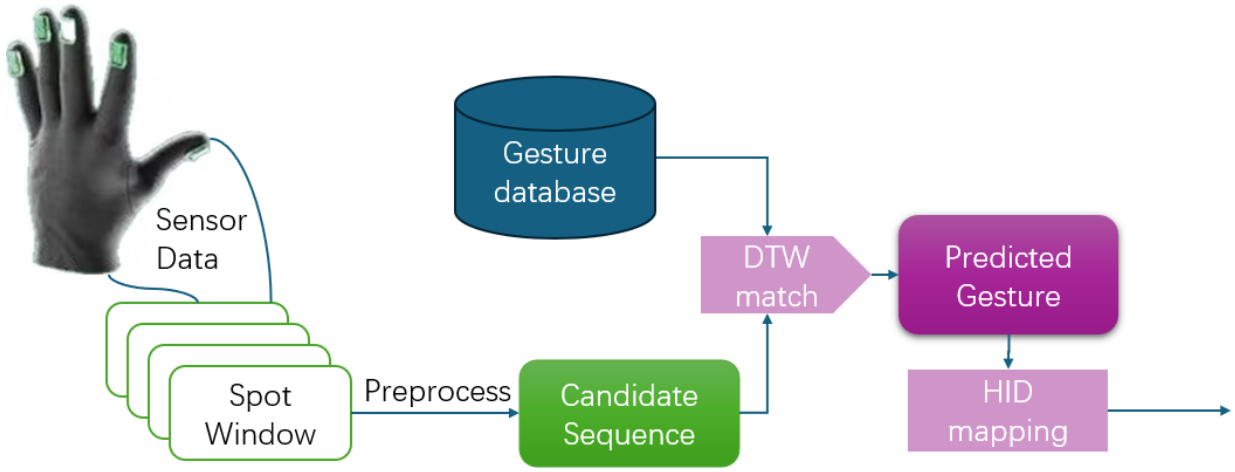

Figure 17: Gesture recognition procedure

Here, we use a time window approach to spot user data and perform gesture recognition and the whole procedure is shown in the figure. For each window, the size of the data we obtain is $[num\_sensors * 6, window\_length]$, which is the data from all sensors multiplied by the window length. Then, we match the candidate sequence with the database using DTW(Dynamic Time Warping) algorithm, and if the minimum cost is below a certain threshold, the gesture is recognized. Given sequence a and b, the DTW algorithm is as follow:

$$\text{DTW}(1,1) = \|a_1 - b_1\|^2,$$

$$\text{DTW}(i,j) = \|a_i - b_j\|^2 + \min\{\text{DTW}(i-1,j),\ \text{DTW}(i,j-1),\ \text{DTW}(i-1,j-1)\}$$

There, the cost will be:

$$\text{Cost} = \text{DTW}(n,m)$$

In addition, to prevent a gesture from being recognized as a sub-gesture (e.g., a double-click being recognized as two single clicks), we simultaneously sample windows of different lengths and take the action recognized by the largest window. Finally, the matched gesture is implemented as a mouse or keyboard shortcut via HID.

3 Requirements and Verification

3.1 Requirements and Verification

The requirements and verifications are shown in the following table.

Requirement	**Verification**
1. Relative Self Test Response (STR) must within 14%	When self-test is activated, the on-board electronics will actuate the appropriate sensor. This actuation will move the sensor's proof masses over a distance equivalent to a pre-defined Coriolis force. Read the output values and compute the bias from factory trim.
2. The mouse movement function is working properly.	(a) The direction of mouse movement and the direction of hand motion can be accurately matched. (b) The mouse movement speed has a linear relationship with the hand movement speed.
3. The accuracy of shortcut key mapping is above 70%.	Given a set of gestures (functions), have the user perform them one by one to measure the accuracy.
4. Battery input voltage must be 3.7 V ±10%	(a) Connect the battery to voltmeter to measure the battery terminal voltage after charging.
5. The charging process should be monitored by the Red and Green LED correctly	(a) Connect partially discharged battery (b) Observe LED4 (Red) on while charging (c) Observe LED5 (Green) on after charge completes
6. Constant current charging should maintain 100 mA ±20%	(a) Apply current meter in circuit to measure current (b) Power system via 5 V USB (c) Ensure charging current remains within 80–120 mA range

Requirement	Verification
7. The voltage output of the converter must be 3.3 V ±10%	(a) Connect oscilloscope to 3.3 V output, and make sure the circuit is powered by the battery (b) Ensure output voltage remains stable in range 2.97–3.63 V load
8. ESP32 should correctly receive data from Micro USB interface	(a) The program can be correctly flashed into the ESP32.
9. ESP32 should correctly receive data from MPU sensor	(a) Angular velocity and linear acceleration data could be read in the monitor.

3.2 Tolerance Analysis

3.2.1 IMU Based Gesture Sensing System

The MPU6050 is capable of configuring up to two I2C addresses, indicating that each I2C bus can support two IMUs. However, the design requires a total of six IMUs: Five are positioned at the fingertips, and one is mounted to the motherboard to provide the orientation of the hand. To address this challenge, the utilization of an I2C MUX, such as the TCA9548A, is recommended. This device is an 8-to-1 I2C MUX, which receives eight I2C SDA and SCL inputs and two selection bits, thereby enabling the utilization of up to sixteen I2C devices with a common address on two I2C buses.
A further inquiry pertains to the fresh rate of the gesture sensing system. The fresh rate dictates the number of samples per second, which exerts a substantial influence on the efficacy of gesture recognition. The MPU6050 generates 6 16-bit values, corresponding to 3 linear accelerations and 3 angular accelerations. Notwithstanding the raw data overhead, three additional bytes of data are necessary for the I2C protocol: the start signal, ACK, and end signal. Hence, one MPU6050 requries

$$6 \times 2 + 3 = 15 \text{ bytes per transmission} \tag{1}$$

The I2C bus has been demonstrated to facilitate data transmission at a rate of up to 400 kilobits per second. Consequently, the 6 MPU6050 is capable of sampling the gesture at a maximum rate give as follows,

$$\begin{aligned} f_{max,sampling} &= \frac{Bandwidth}{DataPerTX} \\ &= \frac{400,000}{6 \times 15 \times 8} \\ &\approx 555Hz \end{aligned} \tag{2}$$

which is sufficient for high precision gesture recognition.

3.2.2 Gesture Recognition System

For the gesture recognition model, the accuracy of the model is inevitably influenced by the dataset. To evaluate the model's performance, detect gesture recognition errors, and identify failures, we use classic classification model evaluation metrics, including Accuracy, Precision, Recall, and AUC-ROC curve. In addition, to prevent a gesture from being recognized as a sub-gesture (e.g., a double-click being recognized as two single clicks), we simultaneously sample windows of different lengths and take the action recognized by the largest window. As for the inference speed, we aim to keep the dataset size within 24kB, and typically, gesture mapping can be completed within 50ms to 200ms.

3.2.3 Communication System

- Theoretical Speed
  BLE 5.0 **2 Mbps PHY** theoretical maximum speed: **2 Mbps = 250 KB/s**. However, BLE has **protocol overhead**, and the actual data rate is much lower than the theoretical value.
- Key Factors Affecting Actual Speed
  BLE 5.0 data transfer is influenced by **Maximum Transmission Unit (MTU)**, **connection interval**, and **number of packets per interval**:
  - **MTU (Maximum Transmission Unit):** ESP32 BLE default supports up to 247 bytes.
  - **Connection Interval:** Minimum 7.5 ms (typically set to 10 ms).
  - **Maximum 6 packets per interval** (per BLE specification).

Effective payload per data packet:

- MTU = 247 bytes (excluding BLE headers)
- L2CAP/ATT overhead = 7 bytes
- GATT payload = 247 - 7 = 240 bytes

Maximum Data Transfer Per Connection Interval

- One packet = 240 bytes
- Max 6 packets per interval
- Connection interval = 10 ms (optimal setting)

Maximum BLE 5.0 throughput calculation:

$$240 \times 6 = 1440 \text{ bytes/10 ms} = 144 \text{ KB/s} = 1.15 \text{ Mbps}$$

If the connection interval is set to 7.5 ms, throughput can be increased, but power consumption will rise. Actual BLE 5.0 2 Mbps mode throughput is approximately 1.4 Mbps (175 KB/s), lower than the theoretical 2 Mbps.

Final Choice of UART Baud Rate: 1.5 Mbps (1500000 bps). This Baud Rate ensures stable BLE 2 Mbps mode throughput of 1.4 Mbps.

3.2.4 Power Management System

The selected lithium battery operates at a standard discharge rate of 0.2C to 0.5C, with a maximum discharge rate of 1C, corresponding to 0.3 A for a 300 mAh cell. To ensure safe current delivery, all 3.3 V and 5 V power traces are routed with a width of 15 mil, which supports up to 1.2 A of continuous current under standard PCB conditions.

All current-limiting resistors and components on the power path have been carefully selected and analyzed for current and voltage stress, ensuring sufficient power tolerance and reliability under expected load conditions.

The combined operating current of key components, including the ESP32, MPU6050, and other peripheral devices, remains below 100 mA. This is well within the battery's maximum rated output (300 mA), and also within the safe current handling capacity of the power traces and all associated circuit elements.

3.3 Safety and Ethics

Our project adheres to ethical and safety principles as outlined in the IEEE and ACM Codes of Ethics[6]. Ethically, we ensure that our design prioritizes reliability, transparency, and public welfare. We commit to honest documentation, respect for intellectual property, and responsible engineering practices. To prevent misuse, we will implement secure communication protocols and adhere to privacy standards to protect data integrity.

This project does not involve high-voltage electricity, but does include lithium batteries. We will select a small 300mAh lithium battery and have read and fully understand the Safe Battery Usage document. In addition, we will apply fuses, battery protection modules to enhance safety. We will strictly follow national and campus safety regulations to ensure that the experiment is conducted in a controlled environment.[7]

4 Cost and Schedule

4.1 Cost Analysis

The following table summarizes the complete component cost for the project, including both standard and custom parts. Prices are listed in Chinese Yuan (RMB, ¥) and converted to US Dollars (USD, $) at an exchange rate of 1 USD = 7.2 RMB.

Description	Manufacturer	Part #	Qty	Total (¥)	Total ($)
Slide Switch	XKB	SK-3296S-01-L1	20	6.90	0.96
BJT Transistor	CJ	SS8050	50	4.47	0.62
FFC Connector	JS	JS10B-05P	20	12.69	1.76
Pin Header	XFCN	PZ254V	50	5.19	0.72
USB-UART Chip	WCH	CH340C	5	17.35	2.41
Pin Header	HANBO	HB-PH3	20	2.56	0.36
Boost IC	HTMX	MT3608	10	3.99	0.55
LDO Regulator	HXY	662K	50	5.38	0.75
Wafer Header	XUNPU	PH2.0-2PWZ	50	3.72	0.52
TVS Diode	TaiHai	nSMD012	20	4.72	0.66
USB Socket	SHOU HAN	MicroXNJ	10	2.45	0.34
Inductor	cjiang	FTC252012S100MBCA	10	2.66	0.37
Charge Protection IC	INJOINIC	IP3005A	5	4.68	0.65
Charging IC	TOPPOWER	TP4056	5	5.73	0.80
Resistor 10k	UNI-ROYAL	0805W8F1002T5E	100	1.15	0.16
Capacitor 0.1uF	SAMSUNG	CL21B103KBANNNC	50	2.07	0.29
Capacitor 0.1uF	SAMSUNG	CL05B104KB54PNC	100	3.12	0.43
FPC Connector	HCTL	HC-FPC-05-09	35	17.90	2.49
Capacitor 2.2nF	SAMSUNG	CL21C222JBFNNNE	20	4.20	0.58
Resistor 4.7k	UNI-ROYAL	0603WAF4701T5E	50	0.28	0.04
PCB Board A	Custom	–	1	79.37	11.02
PCB Board B	Custom	–	1	112.40	15.61
PCB Board C	Custom	–	1	28.00	3.89

Description	Manufacturer	Part #	Qty	Total (¥)	Total ($)
PCB Board D	Custom	–	1	53.64	7.45
ESP32 Module	Espressif	ESP32	3	69.00	9.58
MPU6050 Sensor	InvenSense	MPU6050	8	200.00	27.78
Battery (300mAh)	Generic	Li-ion	3	23.40	3.25
Screw Set	Generic	–	1	3.00	0.42
TCA9548A	TI	TCA9548APWR	4	4.08	0.57

Total Parts Cost: ¥684.10 $95.03

Labor Cost: 4 people $\times$ 80 hours $\times$ ¥80/hour $\times$ 2.5 = ¥64,000.00 $8,888.89

Grand Total: ¥64,684.10 $8,983.90

Note: All the components are purchased online, so the shop labor hour will not be taken into consideration.

4.2 Project Schedule

Date	Jinhao Zhang (EE)	Hongwei Dong (ECE)	Shanbin Sun (ECE)	Zhan Shi (EE)
Mar. 20–24	Design initial power schematic.	Bluetooth test. Define data protocol.	Collect gesture samples. Test IMU stability.	Design IMU-ESP32 I2C connection. Draft PCB layout.
Mar. 25–31	Design initial ESP32 board schematic.	Build preprocessing pipeline (filter, normalize).	Collect standardized gesture set.	Model battery shell. Draft glove appearance.
Apr. 1–7	Choose specialized chip and implement PCB	Train gesture model. Test Bluetooth transmission.	Build I2C bus test routine. UI wireframe.	Develop IMU-ESP32 driver. First PCB test.
Apr. 8–14	Order PCB, Soldering	Improve model accuracy. Integrate with device.	Improve UI. Add custom gesture mapping.	Revise PCB and re-order. 3D print glove.
Apr. 15–21	Test all PCB boards (leave room for possible revision)	Deploy model to ESP32. Map gestures to shortcuts.	UI + recognition integration test.	Assemble glove. Polish design and fit.
Apr. 22–24	Test all PCB boards (leave room for possible revision)	Write deployment scripts. Clean up code.	UI testing and bug fixing.	Finalize exterior. Prepare for demo.
Apr. 25-May. 1	Combine all the subsystems and do final verifications. All together			
May. 2-8	Prepare for mock demo and start writing final report draft. All together			
May. 9-15	Prepare for final demo and the final report. All together			

References

[1] Alex Roney Mathew, Aayad Al Hajj, and Ahmed Al Abri. "Human-computer interaction (hci): An overview". In: *2011 IEEE international conference on computer science and automation engineering*. Vol. 1. IEEE. 2011, pp. 99–100.

[2] W Zhang et al. "Survey of dynamic hand gesture understanding and interaction". In: *J. Softw* 32.10 (2021), pp. 3051–3067.

[3] Encyclopædia Britannica. *A laptop computer*. 2025, Mar 14. URL: %5Curl%7Bhttps://www.britannica.com/technology/computer#/media/1/130429/231796%7D.

[4] Manus Meta. *Quantum Metagloves*. https://www.manus-meta.com/products/quantum-metagloves. 2025, Mar 14.

[5] Electronic Cats. *MPU6050A*. https://github.com/ElectronicCats/mpu6050a. 2025, Mar 14.

[6] IEEE. *"IEEE Code of Ethics"*. 2016. URL: https://www.ieee.org/about/corporate/governance/p7-8.html (visited on 02/08/2020).

[7] University of Illinois at Urbana-Champaign. *Safe Practice for Lead Acid and Lithium Batteries*. Online. Available: https://courses.grainger.illinois.edu/ece445zjui/documents/GeneralBatterySafety.pdf [Accessed: Feb. 28, 2024]. 2016.