ECE 445

SENIOR DESIGN LABORATORY

FINAL REPORT

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

May 13, 2025

Abstract

This project presents the design and implementation of a smart glove system for intuitive human-computer interaction (HCI), offering a wearable alternative to traditional input devices. The glove integrates six ICM-42688-P inertial measurement units (IMUs) to capture finger and hand movements, paired with an ESP32-S3 microcontroller for real-time gesture processing. Hardware subsystems include a compact power management unit with Li-ion battery charging, dual communication via USB and Bluetooth Low Energy (BLE), and a custom PCB for sensor integration. Software components feature SPI-driven IMU communication, a dynamic time warping (DTW) algorithm for gesture recognition, and BLE-based HID profile emulation to map gestures to mouse/keyboard commands. Verification demonstrated over 70% gesture recognition accuracy, real-time response under 0.5 seconds, 5+ hours of wireless operation, and a total device weight below 0.5 kg. The system's modular design supports scalability for AR/VR applications, validating its feasibility as a low-latency, energy-efficient HCI solution.

Keywords: Smart Glove, Human-Computer Interaction (HCI), Inertial Measurement Unit (IMU), Gesture Recognition, ESP32 Microcontroller, Bluetooth Low Energy (BLE), Power Management System, SPI Communication, HID Profile Emulation

Contents

1 Introduction

1.1 Purpose

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, 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]. 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, thereby realizing convenient control of smart devices such as laptops, smartphones, and even AR/VR systems, providing users with a convenient and efficient interactive experience. 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. In our product, we decided to focus on the most basic function - using smart gloves as a ”mouse/keyboard replacement” for laptops. Specifically, the glove can collect information about finger gestures, and map this information to mouse operations such as cursor movement, clicks or scroll wheels, or keyboard input. Although the scope of this project is relatively limited, we fully considered the scalability of the system during design and implementation so that it can be further expanded to more application scenarios in the future.

1.2 Functionality

Considering factors such as product performance and user experience, our smart glove offers the following functionalities:

- The glove can function as a mouse and keyboard to interact with the computer. It allows mouse movement through hand tilting, and maps finger taps to actions such as mouse clicks, double-clicks, and keyboard input.
- The glove supports both wired and Bluetooth connections with a computer. The USB module provides a UART interface, and the ESP32 [3] supports low-power Bluetooth LE. These two connection methods make functionality debugging and user operation more convenient.
- We have implemented battery-powered operation and charging functionality. The power supply section uses a voltage conversion circuit to provide a stable 3.3V output, while the charging section ensures that the glove can be fully charged when not in use, meeting the user's need for long battery life.

1.3 Visual Aid

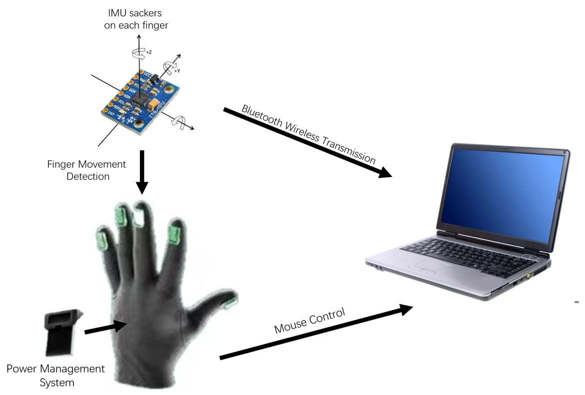

IMU sackers on each finger
Finger Movement Detection
Bluetooth Wireless Transmission
Mouse Control
Power Management System

Figure 1: Visual Aid[4][5][6]

1.4 Subsystem Overview

1. **IMU Based Gesture Sensing System:**The gesture sensing system is responsible for capturing the position and the angle information of the finger tip. Each glove is equipped with six ICM42688 sensors, constituting a 6 DOF IMU. The communication infrastructure between the ESP32 and the ICM42688 is facilitated by an SPI bus. The SPI bus driver plays a pivotal role in regulating the MOSI and MISO wires and implementing a built-in integral and Kalman filtering algorithm.

2. **Gesture Recognition System:** The gesture recognition block uses sensor data for mouse movement and shortcut mapping through a gesture recognition algorithm. We utilize the acceleration along x and z axis to control the mouse. And we use the dynamic time warping algorithm to recognize finger gestures (click). Finally, the command mapping module translates recognized gestures into HID-compatible mouse or keyboard inputs. In addition, we developed a user interface for customizing shortcut mappings.

3. **Communication System:** 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 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.

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 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 5v and 3.3v output voltage to the processor and sensor.

1.5 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 to the mouse pointer on the screen, enabling natural movement. At the same time, gestures such as clicking are used as user commands, mapped to actions like left-click, double-click and keyboard input, ensuring stability and reducing gesture misinterpretation or repeated commands.
2. The local gesture recognition system on the chip utilizes a dynamic time warping 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 0.5 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.

1.6 Block Diagram

There are 4 components in our system, lithium battery and voltage regulator circuit, ICM42688 sensors, ESP32 MCU and the host device. The block diagram of the entire system is shown as follows 2.

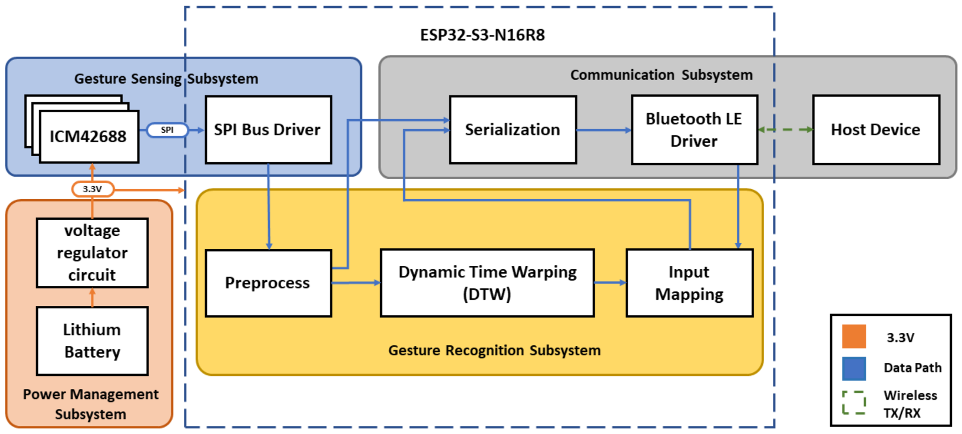


Figure 2: Block Diagram of System Design

2 Design

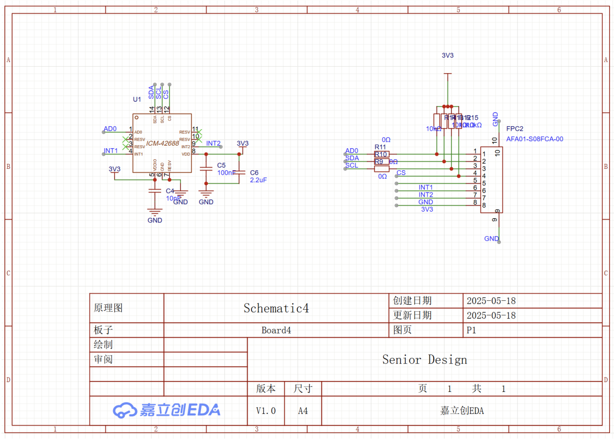
2.1 Hardware Design

2.1.1 ICM-42688-P Sensor Description

The ICM-42688-P [7] is a state-of-the-art six-axis MEMS motion-tracking IC integrating a three-axis gyroscope (configurable full-scale ± 15.6 to ± 2000 /s) and a three-axis accelerometer ($\pm 2/4/8/16\ g$), featuring on-chip digital signal processing that fuses sensor data at sampling rates up to 32 kHz and outputs full-scale 20-bit inertial measurements via SPI (up to 3 MHz) or I^{2}C (up to 1 MHz) interfaces. Housed in a compact 14-pin LGA package ($2.5\,\mathrm{mm} \times 3\,\mathrm{mm} \times 0.98\,\mathrm{mm}$), it draws only $0.88\,\mathrm{mA}$ in low-noise mode, achieving a gyroscope noise density of $2.8\,\mathrm{mdps}/\sqrt{\mathrm{Hz}}$ and an accelerometer noise density of $70\,\mu\mathrm{g}/\sqrt{\mathrm{Hz}}$. These characteristics satisfy the stringent requirements of wearable gesture-recognition gloves, including miniaturization, high energy efficiency, and precise dynamic tracking.

In our glove-based IMU array, six ICM-42688-P sensors are deployed—five on the fingertips and one on the back of the hand. By leveraging the SPI bus, the MOSI, MISO, SCLK, and FSYNC lines (analogous to SDA/SCL/AD0 in I^{2}C mode) are shared across all devices, while each sensor is addressed via its own CS, INT1, and INT2 lines.

In the 14-pin ICM-42688-P package, pin 1 (AD0) selects the I^{2}C address (low = 0x68, high = 0x69); pin 2 (SDA/SDI) functions as the I^{2}C data line or SPI MOSI; pin 3 (SCL/SCLK) serves as the I^{2}C clock or SPI clock input; pin 4 (SDO) is the SPI data-out (MISO) pin; pin 5 (CS) is the SPI chip-select (active low) and must be held high when using I^{2}C; pin 6 (INT1) outputs primary interrupts (e.g., data-ready, FIFO-overflow) to the host controller; pin 7 (INT2/FSYNC/CLKIN) provides a secondary interrupt output, frame-sync input, or external clock input; pin 8 (GND) is ground; pins 9, 11, 12, and 14 are reserved and should remain unconnected; pin 10 (VDD) is the main power supply (1.71–3.6 V) and must be bypassed with 0.1 µF and 2.2 µF capacitors; and pin 13 (VDDIO) is the digital I/O supply (1.71–3.6 V) requiring a 10 nF bypass capacitor.



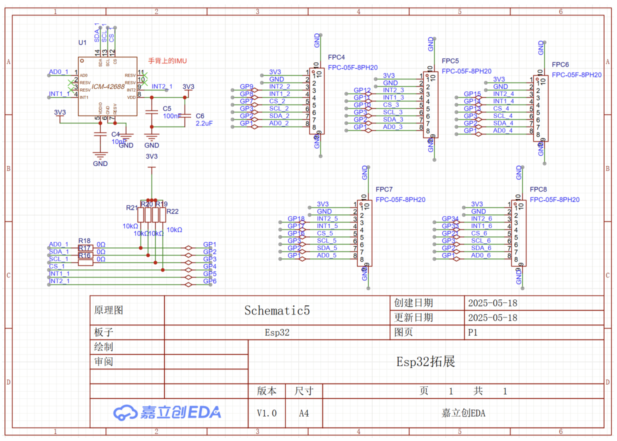


Figure 3: ICM42688 schematic, ESP32 extension Schematic

2.1.2 ESP32 Extension Board Design

Because we chose SPI communication for the ICM-42688, the ESP32 can select which sensor to talk to via each module's CS pin. In our final design we therefore eliminated the external multiplexers and instead developed a custom ESP32 expansion board that routes the ESP32's GPIOs to FPC connectors. Since this PCB sits on the back of the hand, we prioritized an ultra-compact footprint while still accommodating an ICM-42688 IMU to work in concert with the fingertip sensors for capturing relative finger-to-palm positions. Each IMU requires eight signals—SDA, SCL, and AD0 can be shared across all six sensors, while CS, INT1, and INT2 handle individual chip selection and interrupt signalling—resulting in a total requirement of $3 \times 6 + 3 = 21$ GPIO pins. [7]

This expansion board is symmetrically laid out for both left- and right-handed gloves, and leaves numerous unpopulated pads on the back for fly-wire connections to additional glove-mounted peripherals, such as indicator displays.

Our previous design used a full-size ESP32 module that was too large and lacked sufficient I/O, so we switched to the Waveshare ESP32-S3-Tiny development board. The ESP32-S3-Tiny is a highly integrated, miniature MCU board featuring a split-module design that separates the USB and button circuitry to reduce overall PCB thickness and footprint, while still exposing 34 GPIOs. It integrates a dual-core Xtensa® 32-bit LX7 processor running at up to 240 MHz, 512 KB SRAM, 384 KB ROM, 16 KB RTC SRAM, 2 MB PSRAM, 4 MB Flash, Bluetooth LE and 2.4 GHz Wi-Fi, offering excellent RF performance. At just 18×23.5 mm, it perfectly meets our size and performance requirements.[8]

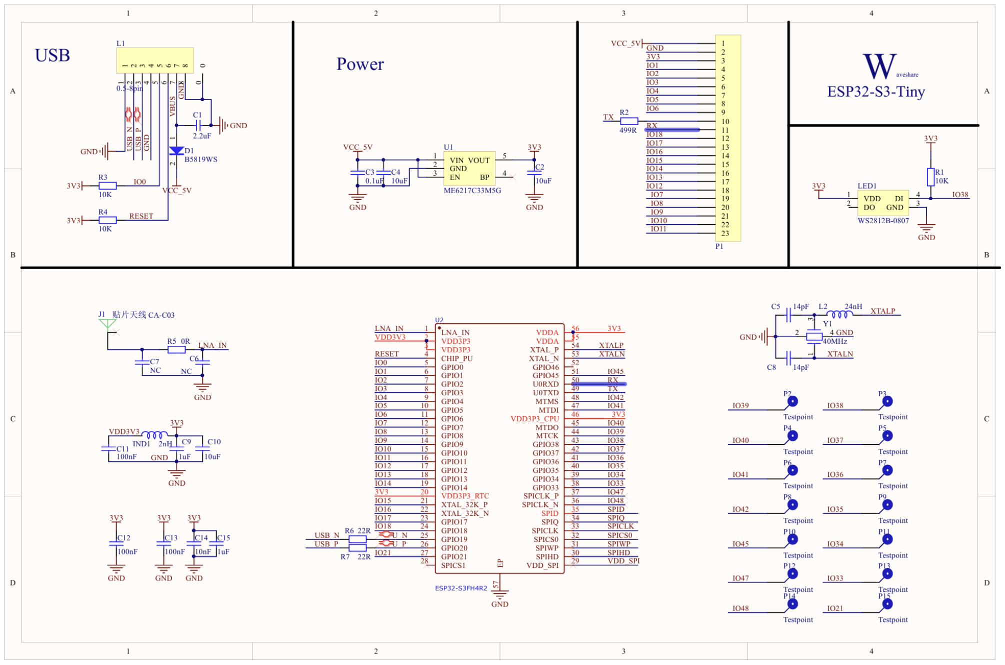


Figure 4: Schematic of Waveshare ESP32-S3 Tiny[8]

2.1.3 Power Management System

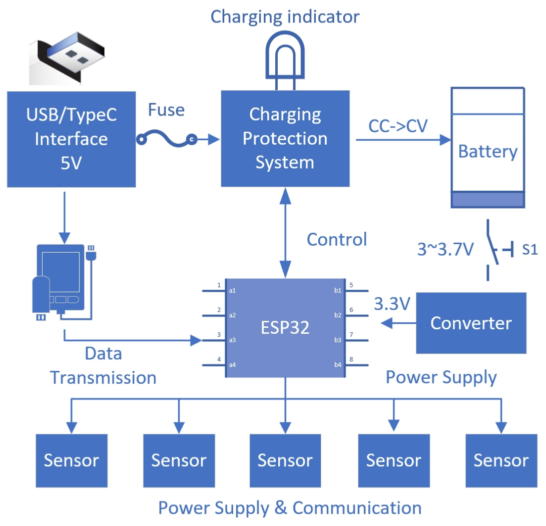


Figure 5: Power Management System Flowchart

As shown in Figure 5, the power management system is divided into two main sections: the battery charging section and the processor power supply section.

1. Battery Charging Section
    - The system receives a 5V power supply via the USB TypeC/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 USB TypeC connector helps simplify later software integration and coding.
2. Processor Power Supply Section
    - The power board first boosts the lithium battery voltage to 5V with a tolerance of ± 1V. This 5V output is then fed into the ESP32 board, where an onboard linear regulator steps it down to 3.3V with a tolerance within 15%.
    - The regulated 3.3V supplies power to the ESP32 as well as its connected peripherals and sensors.

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

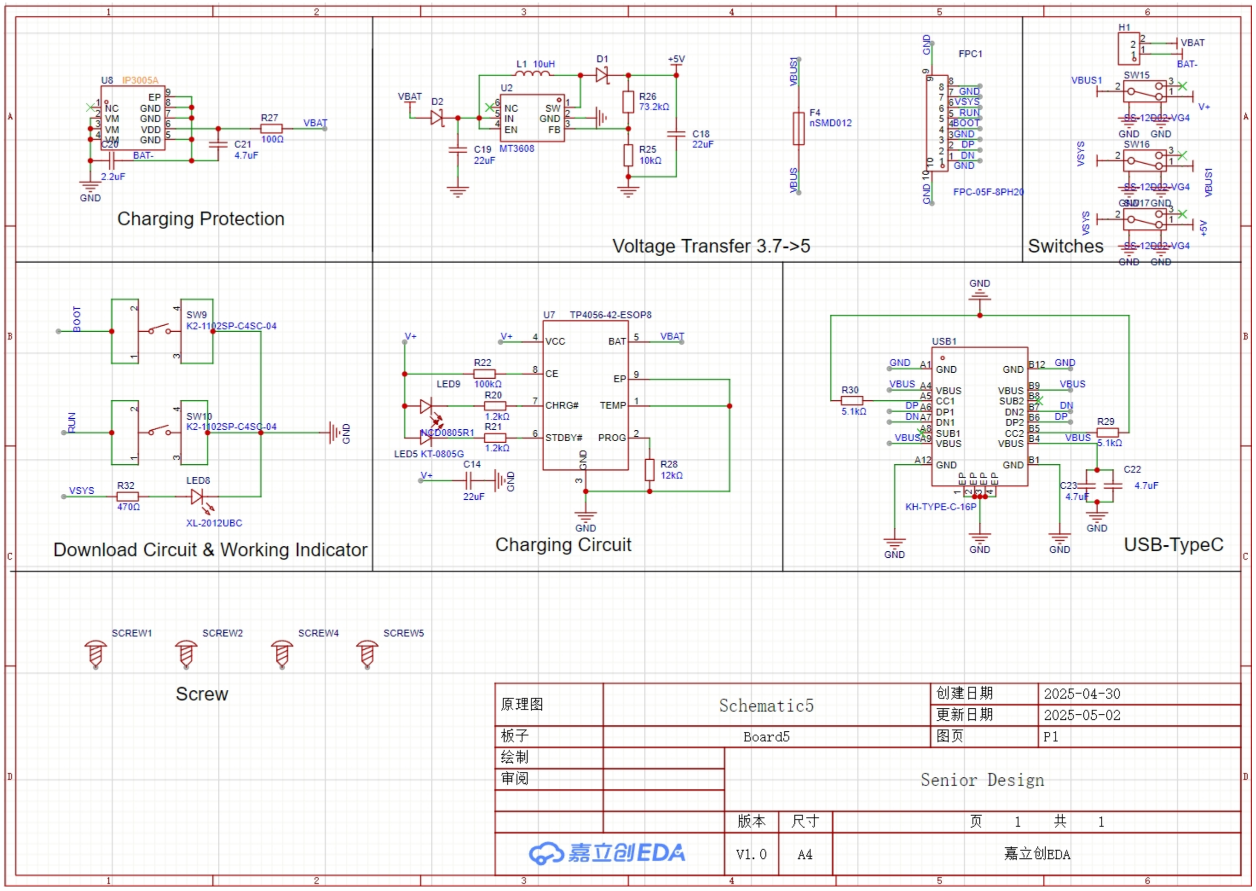


Figure 6: Power board schematic

The power supply board consists of seven functional sections: *Lithium-ion Battery*, *Charging Protection*, *Charging Circuit*, *Voltage Transfer*, *Switches& LED Indicator*, The detailed schematic is included in the main text as Figure 6, while the PCB layout and the assembled board are provided in the appendix (Figures 9, and 15).

Also, to reduce the size of the ESP32 controller board and facilitate battery charging and debugging, the USB download circuit is integrated directly onto the power board. All necessary indicator LEDs—including those for power status, charging in progress, and charging complete—are also incorporated. Additionally, multiple power control switches are provided to flexibly toggle between battery, USB, and charging modes. The power board and lithium battery are housed together in a compact, 3D-printed enclosure designed for wearable use on the wrist, which is shown in the appendix (Figure16).

- **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 (300mAh, as shown in Figure17 in the appendix). 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
  - 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 → 5 V → 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 was implemented on the ESP board, which uses a linear regulator to step down 5 V to 3.3 V.

- **USB TypeC Interface:** The USB 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

- **Download Circuit & Working Indicator:** An onboard USB-to-serial circuit with a built-in reset and boot button allows firmware download to the ESP32 via a single click.

  Additional LEDs indicate system activity and operation status. Specifically, the blue LED indicates that the power supply circuit is functioning properly, the red LED lights up during battery charging, and the green LED turns on when charging is complete. These indicators help monitor system behavior in real time.

- **Switches:** The board includes three control switches for flexible power management:

  - SW15 controls the start and stop of battery charging
  - SW16 toggles between USB Type-C input and 5 V direct power output

- SW17 enables or disables power output from the battery

- **FPC Connector:** An 8-pin, 0.5 mm-pitch FPC connector is used to interface with the ESP32 board, providing both data communication and power supply. This flexible connector simplifies integration and ensures reliable transmission between the main control board and the ESP module.

2.2 Software Design

2.2.1 Overview

Our software design has three main parts, SPI 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 constant expression.

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 SPI driver, IMU driver and bluetooth driver is shown as follows.

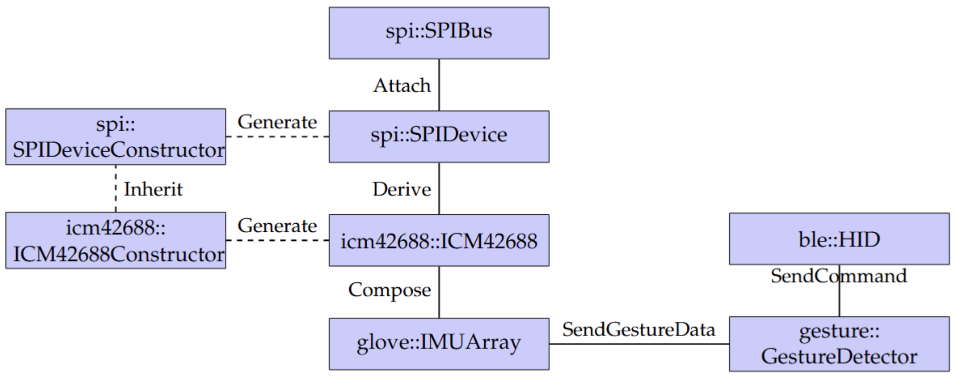


Figure 7: Datapath Flowchart

2.2.2 SPI Driver and IMU Driver

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

1. SPIBus: Represents an SPI bus.
2. SPIBusConstructor: A builder class for creating and configuring an SPIBus instance. Allows customization of parameters like clock source, glitch ignore count, interrupt priority, and internal pull-up resistors.
3. SPIDevice: Represents an SPI 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. SPIDeviceConstructor: A builder class for creating and configuring an SPIDevice instance. Allows customization of parameters like clock speed, chip selection pin, transaction event callbacks. The driver uses ESP-IDF's SPI APIs and ensures error handling .

The icm42688 namespace provides a high-level interface for interacting with the ICM42688 IMU (Inertial Measurement Unit) sensor. It leverages the SPI 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. ICM42688 Class: Encapsulates the functionality of the ICM42688 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. ICM42688Constructor: A builder class for creating and configuring an ICM42688 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 ICM42688 sensor, making it easy to configure and retrieve data while abstracting low-level SPI communication details.

2.2.3 Bluetooth Low Energy Driver

We will be using built-in Bluetooth of ESP32 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 classic Bluetooth, BLE has lower connection delay and lower power consumption while providing large range of Bluetooth advertisement, 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.

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.

2.2.4 Gesture Recognition System

1. **Mouse movement.**

   For the mouse's X-axis and Y-axis movements, we utilized the accelerometer data from the ICM-42688-P mounted on the back of the hand. First, the sensor experiences a downward acceleration due to gravity. We use the x-component of the accelerometer as the horizontal movement speed of the mouse, and the y-component as the vertical movement speed. In this way, tilting the back of the hand left/right or forward/backward can control the mouse movement on the screen.

2. **Data preprocessing and database construction.**

   Unlike simple mouse movement, gesture mapping is achieved through a dynamic mapping model. The data output by the ICM-42688-P 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.

   We define several commonly used mouse and keyboard commands—such as clicking, number inputs 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 $[6, sequence\_length]$, 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.

3. **The process of gesture recognition**

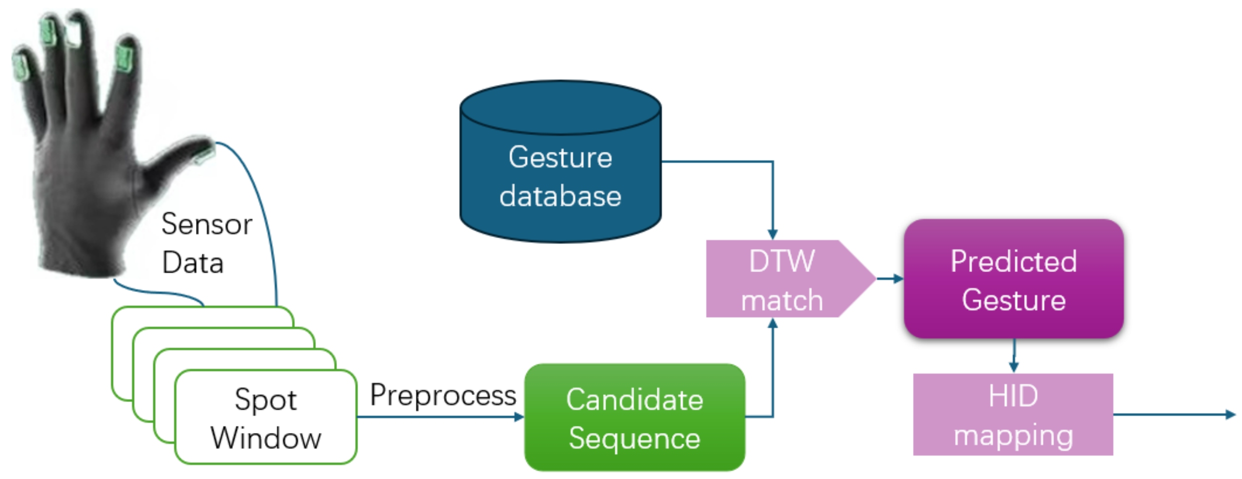


Figure 8: Gesture recognition procedure

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 $[6, window\_length]$, which is the data from one sensor 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 follows:

$$\mathrm{DTW}(i,j) = \begin{cases} \|a_1 - b_1\|^2, & \text{if } i = j = 1 \\ \|a_i - b_j\|^2 + \min \begin{Bmatrix} \mathrm{DTW}(i-1, j), \\ \mathrm{DTW}(i, j-1), \\ \mathrm{DTW}(i-1, j-1) \end{Bmatrix}, & \text{otherwise} \end{cases}$$

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

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. To reduce latency, we map each finger to a single shortcut key (as specified by the user through the interactive interface).

2.3 Design Alternatives

2.3.1 Power Supply Board Design

An earlier version of the power supply and ESP32 extension PCB was fabricated but later discarded and optimized due to several design issues. Precisely, the board had low integration, requiring multiple external connections, and the auto-download circuit was unstable, causing upload failures. Additionally, the layout of LEDs and switches was not user-friendly or visually coherent.

To address these problems, we redesigned the PCB with higher integration. The new version includes a more reliable manual download circuit and arranges indicators and switches more logically. This not only improved functionality but also enhanced the overall appearance and usability. The previous version is shown in Appendix A, Figure18 for reference.

2.3.2 Selection for Different IMUs

The previously used chip MPU-6050 [9], while popular for its integrated DMP and low cost, suffers from several drawbacks: it offers only a 16-bit data output resolution, limiting its sensitivity; its gyroscope noise density is relatively high at $5\,\mathrm{mdps}/\sqrt{\mathrm{Hz}}$ and its accelerometer noise density at $400\,\mu\mathrm{g}/\sqrt{\mathrm{Hz}}$; it supports only I^{2}C communication (100/400 kHz) which can become a bottleneck at higher sampling rates; it consumes $3.6\,\mathrm{mA}$ in active mode (with only a minimal 5 µA sleep current), reducing battery life in continuous-track applications; its QFN-24 package ($4\,\mathrm{mm} \times 4\,\mathrm{mm} \times 0.9\,\mathrm{mm}$) is larger than more modern alternatives; and its built-in temperature compensation guarantees only ±1 % stability over –40 °C to +85 °C, which may be insufficient for precision tracking in fluctuating environments.

Although both devices provide six-axis inertial sensing, the ICM-42688-P markedly outperforms the MPU-6050: it delivers 20-bit output (vs. 16-bit), reduces gyroscope noise density by approximately 44 % (2.8 mdps/$\sqrt{Hz}$ vs. 5 mdps/$\sqrt{Hz}$) and accelerometer noise density by about 82.5 % (70 µg/$\sqrt{Hz}$ vs. 400 µg/$\sqrt{Hz}$), supports high-speed SPI (up to 3 MHz) in addition to I^{2}C (up to 1 MHz) for faster data throughput, draws only 0.88 mA in low-noise mode (vs. 3.6 mA), and comes in a smaller LGA-14 package (2.5 × 3 × 0.98 mm vs. 4 × 4 × 0.9 mm), making it a superior choice for compact, low-power, high-precision wearable gesture-recognition systems.

Product	**ICM-42688[7]**	**MPU-6050[10]**
Manufacturer	TDK InvenSense	TDK InvenSense
Gyroscope Range	± 15.625 to ± 2000 dps	± 250 to ± 2000 dps
Accelerometer Range	± 2 to ± 16 g	± 2 to ± 16 g
Gyroscope Noise	3.5 mdps/$\sqrt{\text{Hz}}$	0.01 dps/$\sqrt{\text{Hz}}$
Accelerometer Noise	70 µg/$\sqrt{\text{Hz}}$	400 µg/$\sqrt{\text{Hz}}$
Interface	I2C, SPI	I2C
Supply Voltage	1.71 to 3.6 V	2.375 to 3.46 V
Power Consumption	0.65 mA (6-axis)	3.8 mA (6-axis)
Package Size	$2.5 \times 3 \times 0.91$ mm	$4 \times 4 \times 0.9$ mm
Features	6-axis (3-axis gyro + 3-axis acceleration), Advanced programmable filtering, Wake-on-motion, Programmable interrupts, on-chip Motion Processing engine for gesture recognition	6-axis (3-axis gyro + 3-axis acceleration), Basic low-pass filtering, Motion detection interrupt, on-chip Digital Motion Processor

Table 1: Comparison of ICM-42688 and MPU6050 IMU Sensors

Based on the above comparison, we choose ICM-42688 as our IMU sensor for the smart glove. The ICM-42688 has a lower zero-rate shift and better performance in terms of noise and power consumption. Meanwhile, smaller package and built-in gesture recognition engine make it more suitable for our application.

3 Requirements and Verification

3.1 Requirements and Verification

The **completeness of requirements** and **appropriate verification procedures** are shown in Appendix C.

3.2 Quantitative Results

3.2.1 Software Design

By declaring functions as consteval, it is ensured that they can only be called and evaluated at compile time, thus eliminating potential runtime costs altogether. For example, complex parameter settings that need to be calculated during SPI driver initialization (e.g., baud rate configuration, data frame format, etc.) can now be done directly at compile time without any runtime support.
In a test scenario, we compare the performance of the SPI driver using traditional runtime computation with that using constexpr and consteval optimizations. The results show that over the course of processing 10,000 standard SPI transactions, the optimized version reduces the average additional latency per transaction to 0 nanoseconds (i.e., there is no additional runtime overhead), while the unoptimized version has an average latency increase of about 50 nanoseconds. At the same time, the compiled assembly code is the same as the assembly code that directly manipulates structures to set up the ESP32's built-in SPI controller, which saves the ESP32's on-chip storage space while bringing about zero runtime overhead

3.2.2 Hardware Design

In accordance with the hardware verification requirements listed in Appendix C, Table 4, we conducted functional validation on both the LED indicators and the regulated power outputs.

The charging process was verified by observing the red LED (LED4) turning on during charging and the green LED (LED5) illuminating upon completion. Power supply status was also correctly indicated by the blue LED when either the battery or USB input was enabled, **as shown in Appendix B, Figure 19.** These results confirm that the LED indication logic operates as expected.

For voltage verification, a digital multimeter was used to measure the regulated outputs. **As shown in Appendix B, Figure 20**, the 5 V output was measured at 5.261 V and the 3.3 V output at 3.2887 V. According to the design specification, the allowed tolerance is $\pm 10\%$, which corresponds to the acceptable ranges of 4.5–5.5 V and 2.97–3.63 V respectively. Both measured values fall well within these limits, thereby confirming that the voltage regulation circuit satisfies the hardware design requirements.

4 Conclusion

4.1 Accomplishments

The smart glove we designed fully replaces both mouse and keyboard functions through intuitive hand and finger gestures. By sensing hand tilt, the glove drives precise cursor movement; by detecting individual finger taps, it generates left-clicks, double-clicks, and customizable keyboard inputs. It supports both wired USB (via CP2102/CH340 UART bridge) and low-energy Bluetooth (BLE) connections on the ESP module, offering flexible, reliable communication and simplifying debugging and user operations. The integrated power system delivers a stable 3.3 V output through a dedicated voltage-conversion circuit, while the built-in charging module allows rapid recharging during idle periods, ensuring extended battery life and uninterrupted use.

4.2 Uncertainties

Despite the system's overall functionality, several limitations remain:

1. **Limited Gesture Mapping.** At present, only simple point-and-tap gestures are supported; more complex motions such as a closed fist, pinch-and-rotate, or multi-finger swipes cannot be reliably detected or mapped to commands. Extending the sensor suite and refining the recognition algorithms will be necessary to accommodate richer gesture vocabularies.

2. **Industrial Design and Mounting.** The current prototype exposes the PCB and electronic components, and the battery is housed in a rudimentary rectangular enclosure. Both the back-of-hand module and the finger units attach via Velcro, which can shift during use and feels unstable. A more polished enclosure design and a secure mounting mechanism (e.g. integrated straps or molded housings) are required to improve aesthetics, durability, and user comfort.

3. **Battery Size and Ergonomics.** The onboard battery remains relatively large and adds significant bulk to the glove, hindering fine motor control and causing user fatigue over extended sessions. Future iterations should focus on higher-energy-density cells or distributed power modules to reduce weight and improve ergonomics.

4.3 Future Work

- **On-device Feedback:** Integrate a small display (e.g. OLED or TFT) on the glove to show in real time which command or gesture is currently active, improving usability and reducing confusion.

- **High-Precision Recognition:** Collect a much larger, more diverse dataset and train a large-scale deep learning model (e.g. Transformer, ConvNet+LSTM) to achieve sub-millimeter accuracy in gesture classification, including complex motions such as pinch-and-rotate and multi-finger sequences.

5 Cost and Schedule

5.1 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.

Category	**Qty (pcs)**	**Total (¥)**	**Total ($)**
PCB Boards (A–F)	6	387.85	52.86
Development Boards & Consumables (ESP32, solder paste, etc.)	—	900.00	125.00
Connectors (FFC/FPC, Type-C, headers)	100+	61.47	8.54
Switches (slide, tactile)	25+	9.74	1.35
ICs (CH340, TP4056, LDO, charging IC)	30+	39.46	5.49
Modules & Sensors (MPU6050, TCA9548A)	12	204.08	28.35
Resistors (470, 5.1k, 10k, 0, etc.)	800+	6.68	0.93
Capacitors (0.1uF, 2.2uF, 4.7uF, etc.)	400+	19.54	2.71
LEDs, Indicators & Misc (green LEDs, buttons, screws)	80+	161.47	22.45
Total Parts Cost	—	**1,790.29**	**248.68**

Table 2: Detailed Summary of All Components (Original + Additional)

Total Parts Cost: ¥1,790.29 $248.68

Labor Cost: 4 people × 120 hours × ¥80/hour × 2.5 = ¥96,000.00 $13,320.00

Grand Total: ¥97,790.29 $13,568.68

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

5.2 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			

6 Ethics

6.1 Ethics and Safety

Our project adheres to the ethical principles set forth by the IEEE and ACM Codes of Ethics[11]. From the outset, we have committed to designing a system that upholds public welfare, avoids harm, and operates with complete transparency. All major design decisions—ranging from sensor selection to communication architectures—are evaluated against these ethical standards to ensure that user safety and societal benefit remain paramount.

We maintain rigorous documentation practices throughout the development lifecycle. Every hardware revision, firmware update, and software algorithm is logged in detail, with clear attributions for any third-party libraries or components. By doing so, we enable reproducibility, facilitate independent review, and foster a culture of responsible engineering. Any discovered limitations or failure modes are reported honestly, along with proposed mitigation strategies.

To prevent misuse and protect user data, our glove's wireless communications are secured using mutually authenticated, encrypted channels. We intentionally collect only the minimal gesture-related telemetry required for accurate classification, and we will not store any personally identifiable information without explicit, revocable user consent. Access controls and secure boot mechanisms further guard against unauthorized firmware tampering.

Although our system operates entirely at low voltage, it incorporates a 300 mAh lithium-ion battery, which carries well-known risks of overcharge, over-discharge, and thermal runaway. We have thoroughly reviewed the UIUC Safe Battery Usage guidelines[12] and will select only cells certified to current industry standards. Each battery pack will include an integrated protection module—guarding against over-current and over-temperature conditions—and a resettable fuse on the main power rail.

Finally, all assembly, testing, and user trials will be conducted under national and campus laboratory safety regulations. Experiments will occur in supervised, controlled environments, with trained personnel, emergency shutdown procedures, and appropriate personal protective equipment in place to ensure that both operators and bystanders remain safe.

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] Espressif. *ESP32-S3 Technical Reference Manual*. Dec. 2024. URL: https://www.espressif.com.cn/sites/default/files/documentation/esp32-s3_technical_reference_manual_en.pdf (visited on 04/18/2025).

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

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

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

[7] InvenSense. *ICM-42688-P Datasheet*. July 2023. URL: https://invensense.tdk.com/download-pdf/icm-42688-p-datasheet/ (visited on 04/16/2025).

[8] Waveshare. *ESP32-S3-Tiny*. Waveshare Wiki, https://www.waveshare.net/wiki/ESP32-S3-Tiny#.E5.8E.9F.E7.90.86.E5.9B.BE. Accessed: 18 May 2025. 2025.

[9] InvenSense. *MPU-6000-Datasheet*. Aug. 2013. URL: https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf (visited on 04/19/2025).

[10] InvenSense. *MPU-6000 and MPU-6050 Register Map and Descriptions*. Aug. 2013. URL: https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Register-Map1.pdf (visited on 04/14/2025).

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

[12] 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].

Appendix A Supplementary Figures for Project Design

A.1 ESP32 Extension Board Supplementary

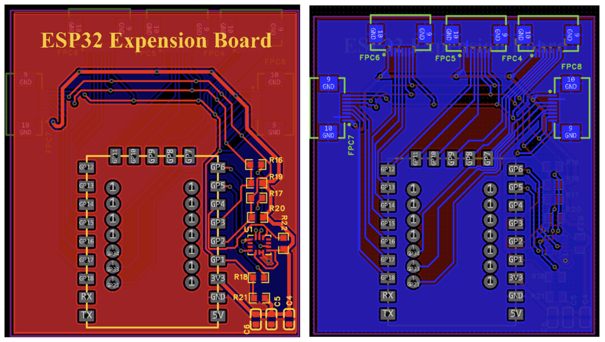


Figure 9: ESP32 Extension Board PCB layout (top and bottom view)

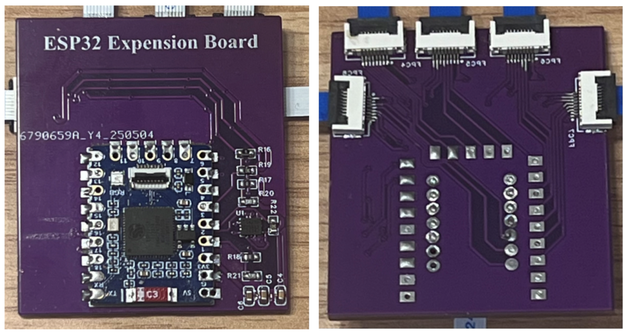


Figure 10: Assembled ESP32 Extension Board

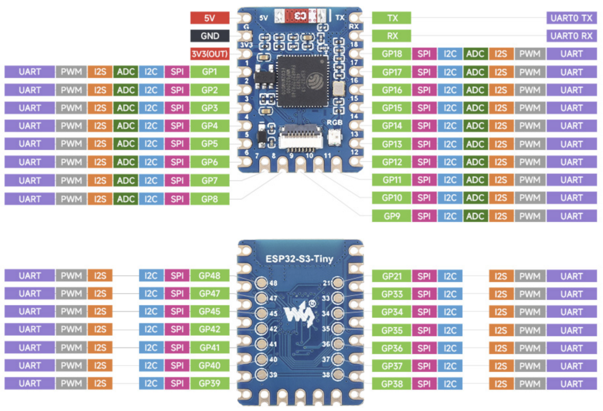


Figure 11: Waveshare ESP32-S3 Tiny[8]

A.2 Sensor Design Supplementary

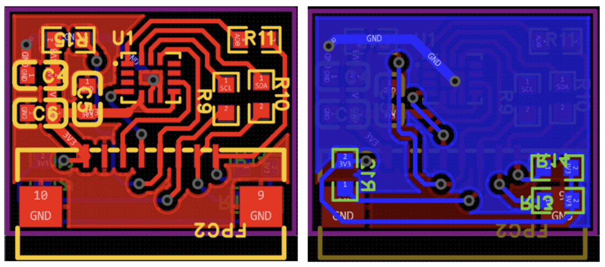


Figure 12: Sensor Board PCB layout (top and bottom view)

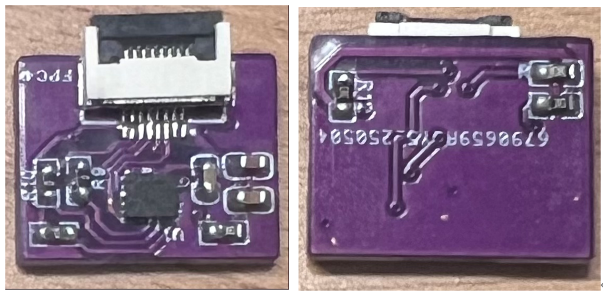


Figure 13: Assembled Sensor Board

A.3 Power Supply Supplementary

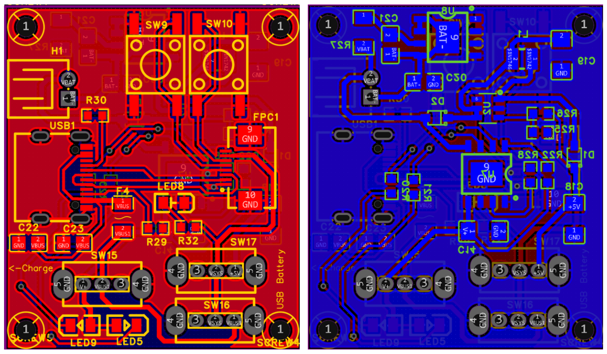


Figure 14: Power board PCB layout (top and bottom view)

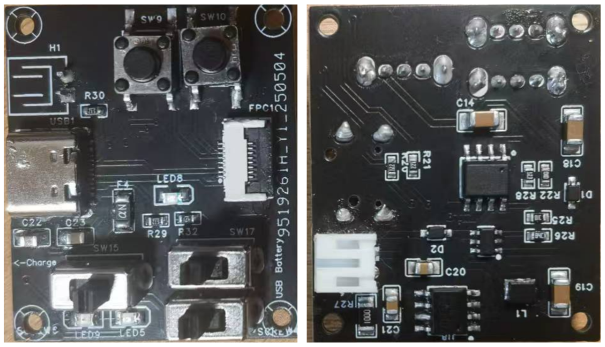


Figure 15: Assembled power board

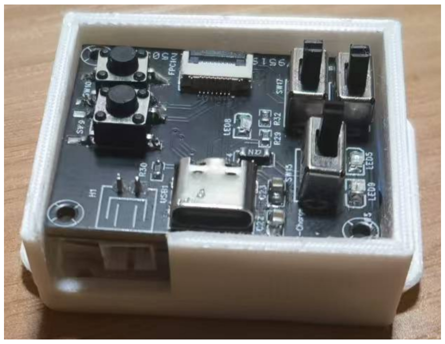


Figure 16: Power "Box" Module

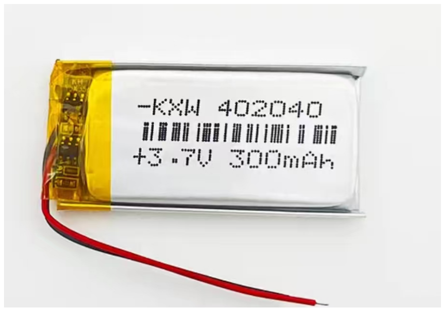


Figure 17: Lithium-ion battery used in the system

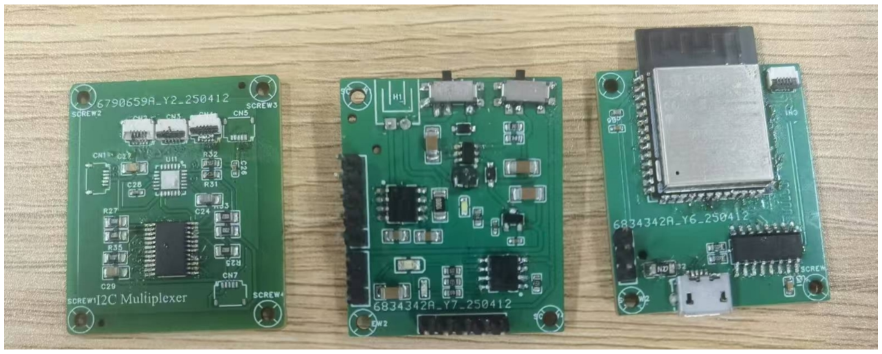


Figure 18: Assembled view of the old ESP32 extension board

Appendix B Supplementary Figures for Verification

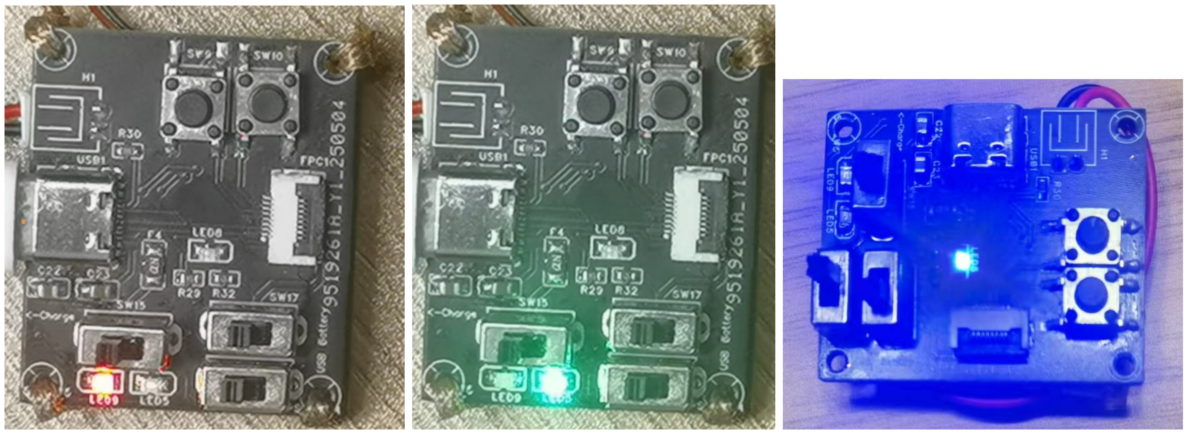


Figure 19: Charging (Red), Charged (Green), and Power (Blue) LED indicators

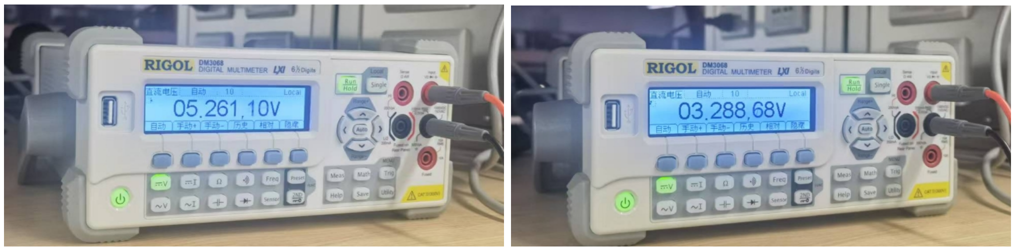


Figure 20: Voltage output measurements: 5.04 V (left), 3.28 V (right)

Appendix C Supplementary Requirements and Verification Table

Category	Requirement	Verification	Pass (Y/N)
Hardware	Charging status should be indicated by red and green LEDs	(a) Connect a partially discharged battery (b) Observe LED4 (Red) during charging (c) Observe LED5 (Green) after charging completes	
	Power supply status should be indicated by the blue LED	(a) When the battery switch is on, the blue LED lights up (b) When the USB switch is on, the blue LED also lights up	
	Voltage outputs should be 5 V and 3.3 V ±10%	(a) Connect a voltmeter to the 5 V and 3.3 V terminals (b) Verify that voltages remain within ±10% of nominal	
Software	Gesture recognition accuracy should exceed 70%	Ask user to perform a series of gesture commands. Measure correct recognition rate across trials.	
	Mouse movement should track hand motion directionally and proportionally	(a) Direction of hand movement matches on-screen pointer (b) Speed of movement shows linear scaling	
	Relative self-test response (STR) must be within 14%	When self-test is activated, the sensor is internally actuated. The output values are read and compared to factory calibration to evaluate STR deviation.	
	ESP32 should receive data from MPU sensor	Monitor angular velocity and linear acceleration data on the serial monitor.	
	ESP32 should receive data via USB interface	Confirm that firmware can be correctly flashed via the Type-C USB port.	

Table 4: Requirements and Verifications Categorized by Hardware and Software