**ELECTRONIC PRODUCT DESIGN PROJECT REPORT**

on

**HAND GESTURE MOUSE**

***Submitted by***

**ABHINAV M BALAKRISHNAN - 20322005**

**ARAVIND P JAIMON – 20322030**

**B RAHUL KRITHIK - 20322036**

**ALEENA A - 20122008**

**APARNA ASOKAN - 20322101**

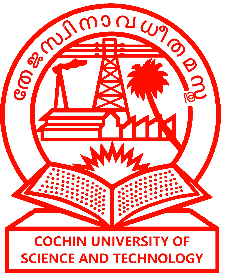
***in partial fulfilment of requirement for the award of the degree***

***of***

**BACHELOR OF TECHNOLOGY**

**in**

**ELECTRONICS AND COMMUNICATION**



**DIVISION OF ELECTRONICS ENGINEERING**

**SCHOOL OF ENGINEERING**

**COCHIN UNIVERSITY OF SCIENCE AND TECHNOLOGY**

**KOCHI-682022**

**APRIL 2025**

|  |  |
| --- | --- |
| DIVISION OF ELECTRONICS ENGINEERING  SCHOOL OF ENGINEERING  COCHIN UNIVERSITY OF SCIENCE AND TECHNOLOGY  KOCHI-682022 | |
| **CERTIFICATE** | |
| *Certified that the seminar report entitled “****HAND GESTURE MOUSE****” is a bonafide work of* ***ABHINAV M BALAKRISHNAN, ARAVIND P JAIMON, B RAHUL KRITHIK, ALEENA A, APARNA ASOKAN*** *towards the partial fulfilment for the award of the degree of B.Tech in Electronics and Communication of Cochin University of Science and Technology, Kochi-682022.* | |
| **Project Coordinator**  Dr. Rinu C Varghese | **Head of the Division**  Dr. Deepa Sankar |

**ACKNOWLEDGEMENT**

We are deeply grateful to God Almighty for granting us the strength and guidance to successfully complete our project. First and foremost, we extend our heartfelt gratitude to Dr. Deepa Sankar, Head of the Division of Electronics and Engineering, for her support and encouragement. We also express our sincere appreciation to our project guides, Mrs. Rinu C Varghese, for their invaluable guidance and for providing us with the necessary resources to accomplish our work.

We would also like to acknowledge the unwavering support and assistance of our lab staff and faculty members from the Electronics and Communication Division, whose contributions were instrumental in the completion of this project.

Lastly, we express our profound thanks to our friends, whose encouragement and help at various stages made this endeavour a success.

**ABSTRACT**

Technology is growing immensely in recent years; certain improvisations need to be made. For people who are tired of the mouse and touchpad, we got something better for them. We are developing a wearable device, which can be used to manipulate and control any system, from a fair distance, without touching the device.

This device, typically worn on the wrist or hand, utilises a module, inbuilt with accelerometer and gyroscope modules to replicate a computer’s mouse functions. The accelerometer is used to measure the orientation and movement of hand in the three-dimensional space, while the gyroscope tracks the rotational motion. Through these sensors, the system can be used to capture simple hand gestures such as rotating, tilting or swiping. These sensors capture real-time motion data from the accelerometer and gyroscope which is then processed by a microcontroller— ESP32 NodeMCU, which maps the specific gestures to basic mouse operations and wirelessly communicates the gesture command to the computer via Bluetooth. Other sensors like flex sensor, push buttons and a capacitive touch sensor are used for clicking and scrolling functions.

This offers not only an intuitive and a hands-free interaction with the computer, but can also offer an alternative method for users with mobility impairments or in situations where traditional input devices are not feasible. This system offers a low-cost, efficient, and intuitive way to interact with devices, particularly useful for individuals with physical disabilities or for applications where touch-based input is impractical. Traditional computer mouse and touchpads require direct physical contact. However, advancements in MEMS (Micro-Electro-Mechanical Systems) sensors and wireless communication enable new ways of interaction, such as gesture-based control. Current input devices are limited by surface dependence and physical constraints.

This project aims to develop a wireless gesture-controlled mouse using an ESP32, in order to read motion data from the MPU6050 and other sensors, and translate it into mouse movements.

**TABLE OF CONTENTS**

[CHAPTER 1 1](#_Toc195979089)

[INTRODUCTION 1](#_Toc195979090)

[CHAPTER 2 3](#_Toc195979091)

[SYSTEM BLOCK DIAGRAM AND EXPLANATION 3](#_Toc195979092)

[2.1 SYSTEM BLOCK DIAGRAM 3](#_Toc195979093)

[2.2 EXPLANATION 4](#_Toc195979094)

[CHAPTER 3 5](#_Toc195979095)

[COMPONENTS 5](#_Toc195979096)

[3.1 NODEMCU ESP32 DEVELOPMENT BOARD 5](#_Toc195979097)

[3.1.1 PIN DIAGRAM 6](#_Toc195979098)

[3.1.2 PIN FUNCTIONS 7](#_Toc195979099)

[3.2 MPU6050 10](#_Toc195979100)

[3.2.1 PIN DIAGRAM 11](#_Toc195979101)

[3.2.2 PIN FUNCTION 12](#_Toc195979102)

[3.3 JUMPER WIRES 14](#_Toc195979103)

[3.4 BREADBOARD 15](#_Toc195979104)

[3.5 TTP223 17](#_Toc195979105)

[3.5.1 TECHNICAL SPECIFICATIONS 18](#_Toc195979106)

[3.5.2 PIN DIAGRAM 18](#_Toc195979107)

[3.5.3 PIN DESCRIPTION 18](#_Toc195979108)

[3.6 PUSH BUTTON 19](#_Toc195979109)

[3.7 FLEX SENSOR 21](#_Toc195979110)

[3.8 RESISTOR 22](#_Toc195979111)

[3.9 RECHARGABLE Li 3.7 V BATTERY 23](#_Toc195979112)

[CHAPTER 4 25](#_Toc195979113)

[CIRCUIT DESIGN AND WORKING 25](#_Toc195979114)

[4.1 CIRCUIT DESIGN 25](#_Toc195979115)

[4.2 PIN CONNECTION 26](#_Toc195979116)

[CHAPTER 5 29](#_Toc195979117)

[TESTING AND RESULTS 29](#_Toc195979118)

[5.1 TESTING 29](#_Toc195979119)

[5.2 BUDGET TABLE 33](#_Toc195979120)

[5.3 RESULTS 33](#_Toc195979121)

[5.4 OUTPUT 35](#_Toc195979122)

[CHAPTER 6 37](#_Toc195979123)

[APPLICATIONS 37](#_Toc195979124)

[CHAPTER 7 42](#_Toc195979125)

[CONCLUSION AND FUTURE PROSPECTS 42](#_Toc195979126)

[7.1 CONCLUSION 42](#_Toc195979127)

[7.1.1 ADVANTAGES 42](#_Toc195979128)

[7.1.2 DISADVANTAGES 42](#_Toc195979129)

[7.1.3 LEARNINGS 43](#_Toc195979130)

[7.2 FUTURE PROSPECTS 43](#_Toc195979131)

[CHAPTER 8 47](#_Toc195979132)

[REFERENCES 47](#_Toc195979133)

[CHAPTER 9 49](#_Toc195979134)

[APPENDICES 49](#_Toc195979135)

**List of Figures**

[Figure 1:Block Diagram 3](#_Toc195971245)

[Figure 2:NodeMCU ESP32 Board 5](#_Toc195971246)

[Figure 3: ESP-WROOM-32 Pin Out 7](#_Toc195971247)

[Figure 4: MPU6050 11](#_Toc195971248)

[Figure 5: MPU6050 Pin Diagram 12](#_Toc195971249)

[Figure 6: Jumper Wires 15](#_Toc195971250)

[Figure 7: Breadboard 16](#_Toc195971251)

[Figure 8: TTP223 17](#_Toc195971252)

[Figure 9: TTP223 Pin Diagram 18](#_Toc195971253)

[Figure 10: Push Button 20](#_Toc195971254)

[Figure 11: Flex Sensor 21](#_Toc195971255)

[Figure 12: 22kΩ resistor 23](#_Toc195971256)

[Figure 13: 3.7 Li Battery 24](#_Toc195971257)

[Figure 14: Circuit Design 25](#_Toc195971258)

[Figure 15: MPU6050 interfacing with ESP32 27](#_Toc195971259)

[Figure 16: Flex Sensor Interfacing with ESP32 28](#_Toc195971260)

[Figure 17: MPU6050 Sensor Values 29](#_Toc195971261)

[Figure 18: Gesture-to-Cursor Mapping 30](#_Toc195971262)

[Figure 19: Click Testing 32](#_Toc195971263)

[Figure 20: Final Connection in Breadboard 35](#_Toc195971264)

[Figure 21: Gesture-to-Cursor Mapping with Clicks 36](#_Toc195971265)

**List of Tables**

[Table 1: GPIO Pin Description 10](#_Toc195971266)

[Table 2: MPU6050 Pin Function 12](#_Toc195971267)

[Table 3: TTP223 Specifications 18](#_Toc195971268)

[Table 4: TTP223 Pin Description 19](#_Toc195971269)

[Table 5: Budget Table 33](#_Toc195971270)

# CHAPTER 1

# INTRODUCTION

In recent years, touchless computing has gained popularity, allowing users to interact with devices through gestures. This project presents a Hand Gesture Mouse, a system that uses an ESP32 microcontroller and an MPU6050 accelerometer-gyroscope sensor module to control a phone’s cursor or computer cursor through Bluetooth communication. The ESP32 reads motion data from the MPU6050 and translates it into mouse movements. The microcontroller interprets specific gestures and maps them to standard mouse operations, such as moving the cursor in x and y direction and scrolling, controlled by hand tilts and directional movements. The processed gesture commands are wirelessly transmitted to the computer using Bluetooth, enabling real-time interaction.

Once the gesture commands are transmitted wirelessly (via Bluetooth) to the computer or smartphone, it detects the ESP32 as a Bluetooth HID (Human Interface Device), similar to a wireless mouse. The operating system (Windows, macOS, Android, etc.) interprets the received signals as standard mouse commands. Cursor movement corresponds to tilt angles from the MPU6050 sensor. The operating system (Windows, macOS, Android, etc.) interprets the received signals as standard mouse commands. The user controls the mouse pointer without touching a surface, making interaction seamless. Adjustments in sensitivity or responsiveness can be made via software calibration.

To ensure a smooth and stable user experience, the Hand Gesture Mouse incorporates various error-handling techniques and stability improvements. The system applies dead zones and filtering techniques to minimize unwanted cursor jitter. One of them is Dead Zone Implementation which is small, unintended hand movements can cause cursor jitter. A dead zone threshold is applied, where minor movements are ignored. Data Filtering (Noise Reduction) that helps to the MPU6050 sensor generate raw motion data, which may contain noise. A simple moving average filter can be used to smooth the sensor readings. This helps eliminate random fluctuations, making the cursor movement more stable. If the Bluetooth connection is lost, the ESP32 can automatically attempt reconnection.

Additional sensors are used for implementing mouse functions and activities like scrolling, clicking and pausing, etc. Sensors like flex sensor and push buttons are implemented for sensing these gestures and it is converted by the ESP32 microcontroller.

The ESP32 enters low-power mode when idle, reducing battery consumption. The device wakes up upon detecting motion, optimizing power efficiency. These enhancements significantly improve cursor stability, accuracy, and user experience, making the system more reliable and practical for daily use Adaptive Sensitivity Adjustment. Users may have different preferences for cursor speed. The system allows for adjustable sensitivity by scaling the gyroscope values.

This mouse, unlike other normal mouse used, doesn’t require a plane surface to work on and can take the advantage of the 3-dimensional space around us, to move the mouse pointer. This prototype can be built in breadboard and later can be designed into a Printed Circuit Board (PCB) in order to reduce space and weight of the built product.

# CHAPTER 2

# SYSTEM BLOCK DIAGRAM AND EXPLANATION

This chapter presents the system block diagram of the Hand Gesture Mouse and explains the role of each component involved. The block diagram provides a high-level overview of how motion is captured, processed, and transmitted to control the cursor. It highlights the interaction between the MPU6050 sensor, ESP32 microcontroller, and the Bluetooth interface for seamless gesture-based control.

## 2.1 SYSTEM BLOCK DIAGRAM

The below figure shows the system block diagram of the hand gesture mouse. It consists of various functional blocks, such as Accelerometer + Gyroscope Module (MPU6050), a Microcontroller (ESP32), a Wireless Module (for Communication), Additional Sensors (Flex Sensor, Push buttons and Capacitive Touch sensor) and a Power Supply.

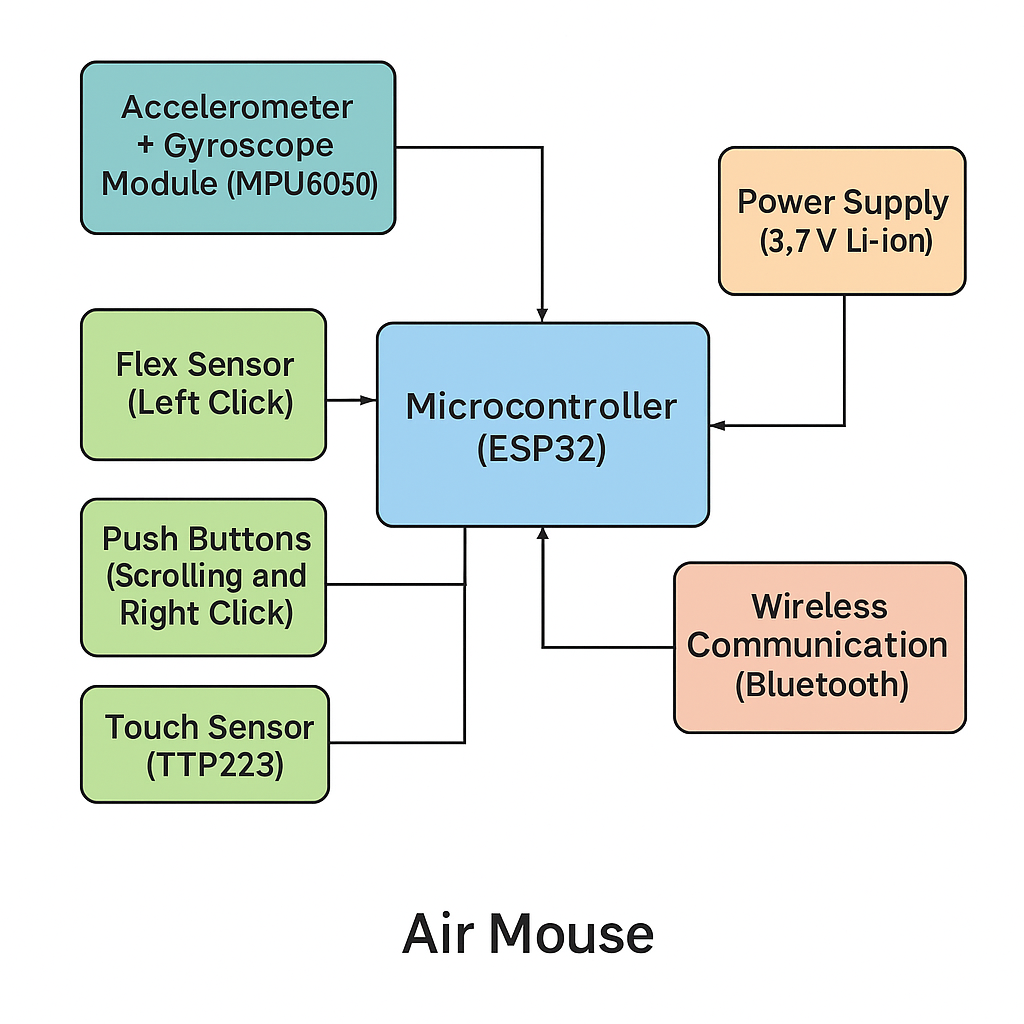


Figure :Block Diagram

## 2.2 **EXPLANATION**

Each of the above components play a crucial role in ensuring smooth and efficient motion tracking, processing, and wireless communication for touchless computing. The Accelerometer + Gyroscope Module (**MPU6050**) is responsible for detecting hand movements and tilt angles. The accelerometer measures linear acceleration, while the gyroscope tracks angular velocity. By combining these two data sources, the module provides precise motion tracking, allowing the system to determine the exact movement and orientation of the user's hand. This motion data is then transmitted to the microcontroller for further processing.

The Microcontroller (**ESP32**) serves as the central processing unit of the system. It receives real-time data from the MPU6050 sensor, processes it, and translates the motion parameters into cursor movements. The ESP32 is a powerful microcontroller with built-in Bluetooth capabilities, making it ideal for wireless gesture-based applications. Its primary function is to interpret the raw motion data from the MPU6050, convert it into meaningful cursor actions, and transmit the necessary commands to the connected device.

Additionally, the microcontroller can manage other sensors, depending on the additional features implemented in the system. A Wireless Module is integrated into the system to establish a communication link between the hand gesture mouse and the target device (such as a computer or smartphone). Since ESP32 comes with built-in Bluetooth, it can be used to wirelessly transmit processed data to the connected device, ensuring real-time cursor control without the need for physical connections.

Bluetooth Low Energy (BLE) is often preferred for such applications due to its low power consumption and stable connectivity. The system also allows for the inclusion of additional sensors, depending on the extra functionalities required. In this case, flex sensor is used to implement left click, push buttons for right click and scrolling and a touch sensor for other gesture controlling. These optional sensors enhance the overall usability of the hand gesture mouse by providing more control features beyond just cursor movement.

The Power Supply is another essential component that ensures the system operates efficiently. The device can be powered by a rechargeable battery or a direct power source, depending on the design preferences. In portable implementations, a lithium-ion battery with a charging circuit is commonly used to provide uninterrupted power for extended use. A wireless module is crucial for the operation of the mouse. It acts as an interface for communication between the human and the device. Bluetooth is a more convenient mode of communication used for these devices, with a good proximity distance and compatibility with other devices.

# CHAPTER 3

# COMPONENTS

## 3.1 NODEMCU ESP32 DEVELOPMENT BOARD

The ESP32 NodeMCU development board is a low-cost, feature-rich microcontroller board based on the ESP32 SoC (System-on-Chip) developed by Espressif Systems. It is widely used in Internet of Things (IoT), wearable tech, home automation, and wireless control projects due to its powerful hardware capabilities and built-in wireless communication.

One of the most significant applications of the ESP32 is in IoT systems, where devices are connected to the internet to exchange data and automate processes. The ESP32’s built-in Wi-Fi and Bluetooth make it an excellent choice for IoT projects, as it allows seamless communication between devices. For example, smart irrigation systems use the ESP32 to monitor soil moisture levels and automatically control water flow based on weather conditions. Similarly, air quality monitoring systems use ESP32 to collect environmental data and send real-time updates to cloud platforms, helping users track pollution levels remotely. The ESP32 is extensively used in home automation systems, where smart devices communicate with each other to improve convenience and efficiency. Smart lighting systems can be built using ESP32, allowing users to control lights remotely using a mobile app or voice commands.

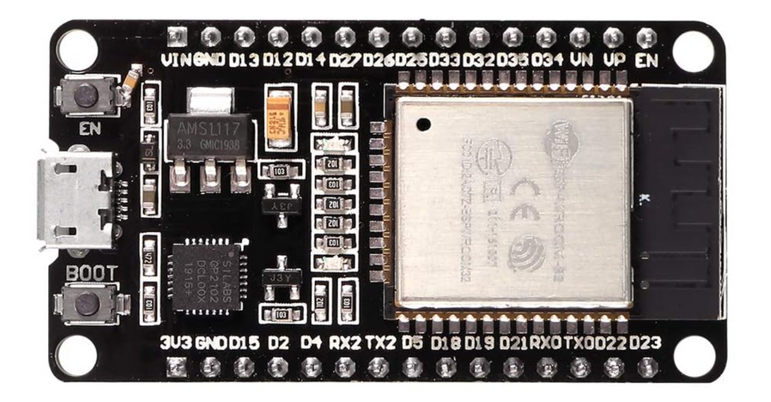


Figure :NodeMCU ESP32 Board

Additionally, smart locks and security cameras powered by ESP32 enable users to monitor and control access to their homes from anywhere in the world. Integration with voice assistants like Amazon Alexa and Google Assistant makes the ESP32 a key component in creating voice-controlled home automation systems. With its support for both Wi-Fi and Bluetooth Low Energy (BLE), the ESP32 is widely used in wireless communication applications. It can serve as a Wi-Fi access point or station, allowing multiple devices to connect and exchange data. This makes it ideal for remote data logging systems, where sensors collect data and send it to cloud servers for analysis.

The ESP32 is also used in smart wearables, where BLE is utilized for low-power communication between a device and a smartphone. The ESP32 is frequently used in embedded systems due to its high processing power and multiple GPIO (General Purpose Input/Output) pins. It is employed in robotics, automation, and industrial control systems. In manufacturing industries, ESP32-based real-time monitoring systems help track machine performance, temperature, and humidity, ensuring smooth operations.

Moreover, automated vending machines use ESP32 for wireless payments and inventory tracking. With its low power consumption and BLE connectivity, the ESP32 is widely used in wearable technology. Fitness trackers and health monitoring devices utilize ESP32 to collect heart rate, body temperature, and step count data. These devices can transmit data to smartphones or cloud platforms for further analysis. The ESP32 is also used in smart glasses and augmented reality (AR) headsets, where it processes sensor data and provides wireless connectivity.

The ESP32 is a versatile microcontroller that has revolutionized multiple industries, from IoT and home automation to embedded systems and wearable technology. Its built-in wireless communication, low power consumption, and high processing capability make it a preferred choice for developing innovative and efficient smart devices. As technology continues to advance, the ESP32 is expected to play a crucial role in shaping the future of connected systems and automation.

The ESP32 microcontroller has 38 or more GPIO pins (depending on the board version), and each pin has multiple functions. These pins are categorized into different types based on their capabilities:

### 3.1.1 PIN DIAGRAM

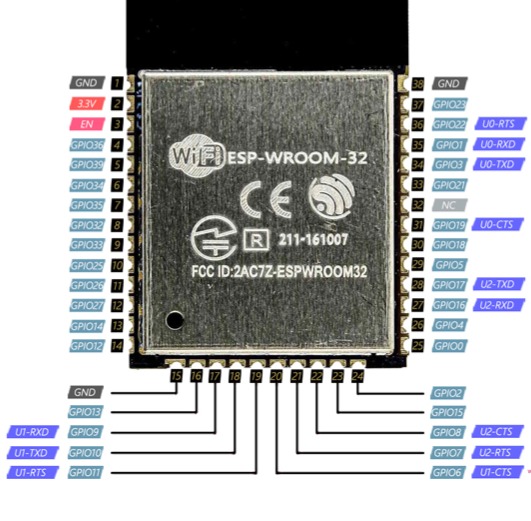


Figure : ESP-WROOM-32 Pin Out

### 3.1.2 PIN FUNCTIONS

1. **Power Pins:** The ESP32 operates at 3.3V, and its power pins include:
   * 3V3 (3.3V Output): Provides power for external components such as sensors and modules.
   * GND (Ground): Completes the electrical circuit and provides a common ground reference.
   * VIN (Voltage Input): Used to supply power to the ESP32 (typically 5V from a USB or external power source).
2. **Digital GPIO Pins**

The ESP32 has multiple GPIO pins, which can function as input or output to control LEDs, motors, or receive signals from sensors. Some key features include:

* + Configurable as input or output based on user needs.
  + Some GPIOs support internal pull-up or pull-down resistors.
  + Can generate PWM (Pulse Width Modulation) signals for dimming LEDs or controlling motor speed

1. **ADC Pins (ADC - Analog to Digital Converter)**

ESP32 has multiple ADC (Analog to Digital Converter) pins, which can read analog signals from sensors like temperature, light, or potentiometers. The ESP32 ADC can read voltages between 0V and 3.3V and convert them into digital values. Some of the commonly used ADC pins include GPIO 32, 33, 34, 35, 36, and 39. Unlike Arduino, the ESP32 has 12-bit ADC resolution, meaning it can measure 4096 different voltage levels.

1. **PWM Pins (Pulse Width Modulation)**

PWM is used for controlling brightness of LEDs or motor speed. All ESP32 GPIO pins can generate PWM signals. The PWM frequency and duty cycle can be adjusted using software. Used in servo motors, LED dimming, and audio signal generation.

1. **Touch Sensor Pins**

The ESP32 includes capacitive touch sensor pins, allowing touch-based input without mechanical buttons. Touch pins include GPIO 0, 2, 4, 12, 13, 14, 15, 27, 32, and 33. Used for touch-based switches, proximity detection, and smart home automation.

1. **I2C (Inter-Integrated Circuit) Pins**

I2C communication is used to connect ESP32 with multiple sensors and modules using just two pins:

* + SDA (Serial Data) - GPIO 21
  + SCL (Serial Clock) - GPIO 22

Used in temperature sensors, OLED displays, and MPU6050 motion sensors

1. **SPI (Serial Peripheral Interface) Pins**

SPI is another communication protocol used for fast data transfer between ESP32 and devices like SD cards, displays, and flash memory.

Default SPI pins:

* + MISO (Master In Slave Out) - GPIO 19
  + MOSI (Master Out Slave In) - GPIO 23
  + SCK (Clock) - GPIO 18
  + CS (Chip Select) - GPIO 5

1. **UART (Universal Asynchronous Receiver/Transmitter) Pins**

UART is used for serial communication, such as debugging and data exchange with computers or other microcontrollers. ESP32 has three UART ports: UART0, UART1, and UART2.Common pins used:

* TX (Transmit) - GPIO 1
* RX (Receive) - GPIO 3

Used in Bluetooth communication, GPS modules, and serial debugging.

1. **DAC (Digital to Analog Converter) Pins**

Unlike many microcontrollers, ESP32 has built-in DAC pins, which convert digital values to analog voltages. DAC pins: GPIO 25 and GPIO 26. Used for audio signal generation and voltage-based control systems.

|  |  |
| --- | --- |
| **GPIO Pins** | **Function** |
| GPIO 0 | -UART0\_TX, ADC, PWM, I2C, SPI, Flash, Boot mode (used for entering serial programming mode) |
| GPIO 1 | - UART0\_RX, ADC, PWM, I2C, SPI, Flash |
| GPIO 2 | - UART1\_RX, ADC, PWM, I2C, SPI, Touch Sensor (Capacitive Touch) |
| GPIO 3 | - UART0\_RX, ADC, PWM, I2C, SPI |
| GPIO 4, 5 | - SPI, I2C, ADC, PWM, Touch Sensor, UART2\_TX, HSPI |
| GPIO 6,7 | - SPI (SDIO), Flash, CLK |
| GPIO 8, 9, 10, 11 | - SPI (SDIO), Flash |
| GPIO 12 | - SPI, ADC, PWM, Touch Sensor, HSPI, SDIO, DAC |
| GPIO 13 | - SPI, ADC, PWM, Touch Sensor, HSPI, DAC |
| GPIO 14 | - SPI, ADC, PWM, Touch Sensor, HSPI |
| GPIO 15 | - SPI, ADC, PWM, HSPI |
| GPIO 16 | - UART2\_RX, ADC, PWM, I2C, SPI |
| GPIO 17 | - UART2\_TX, ADC, PWM, I2C, SPI |
| GPIO 18, 19 | - SPI, I2S, PWM, ADC |
| GPIO 20, 21 | - I2C, PWM, ADC, Touch Sensor |
| GPIO 22 | - UART2\_RX, I2C, PWM, ADC, Touch Sensor, HSPI, I2S |
| GPIO 23 | - SPI, PWM, I2S, ADC, HSPI |
| GPIO 24 | - SPI, PWM, I2S, ADC |
| GPIO 25 | - ADC, PWM, DAC, I2C |
| GPIO 26 | - DAC, ADC, PWM, I2S |
| GPIO 27, 30, 31 | - ADC, PWM |
| GPIO 26 | - SPI, PWM, ADC |

Table : GPIO Pin Description

Some pins have specific functions that are critical during booting or programming the ESP32 (e.g., GPIO0 for boot mode selection), and using them for other purposes might interfere with normal operation. Always check the ESP32 documentation specific to your development board and application.

## 3.2 MPU6050

The MPU6050 plays a crucial role in gesture-based mouse control by detecting hand movements and converting them into cursor movements on a screen. This is achieved using its accelerometer and gyroscope, which measure motion and rotation in real-time. The MPU6050 is a widely used Micro-Electro-Mechanical System (MEMS) sensor that combines a 3-axis accelerometer and a 3-axis gyroscope in a single chip.

Developed by InvenSense, this sensor is extensively used in motion tracking, robotics, gaming, drones, and wearable devices. It provides precise motion data by measuring acceleration and angular velocity, making it a crucial component in applications that require motion sensing and control. The MPU6050 is a 6-degree-of-freedom (6-DOF) sensor, meaning it can detect motion in six different directions. The accelerometer measures linear acceleration along the X, Y, and Z axes, while the gyroscope detects rotational movement along the same three axes. This enables the MPU6050 to track both tilt and angular motion, making it ideal for gesture-based applications. One of the most notable features of the MPU6050 is its Digital Motion Processor (DMP), which processes sensor data internally and reduces the computational load on the microcontroller. The I2C (Inter-Integrated Circuit) communication protocol is used for easy interfacing with microcontrollers like ESP32, Arduino, and Raspberry Pi.

The sensor operates at 3.3V or 5V, making it compatible with various embedded systems. The MPU6050 uses microfabricated capacitive sensing technology for motion detection. The accelerometer determines acceleration by measuring changes in capacitance due to movement, while the gyroscope measures angular velocity based on the Coriolis effect. The data from both sensors can be combined using sensor fusion algorithms to provide highly accurate orientation tracking. The DMP processes this raw data, filters out noise, and provides refined motion values. Due to its high accuracy, compact size, and affordability, the MPU6050 is widely used in various fields.

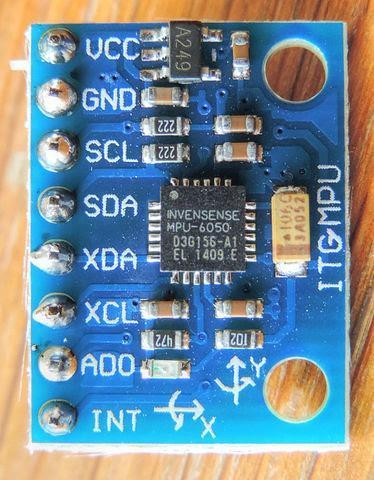


Figure : MPU6050

The MPU6050 module typically has eight pins, each serving a specific function. These pins allow the sensor to interface with microcontrollers like ESP32, Arduino, and Raspberry Pi via I2C communication. The following are the primary pins of the MPU6050:

### 3.2.1 PIN DIAGRAM

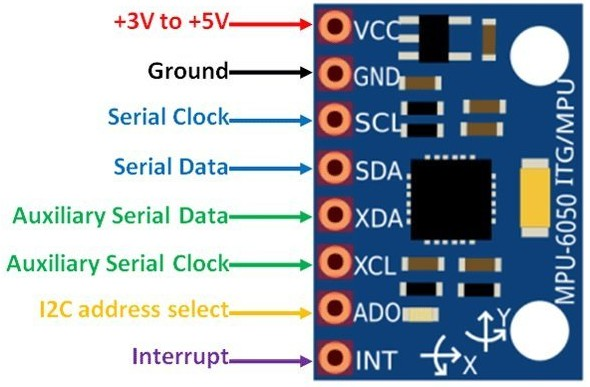


Figure : MPU6050 Pin Diagram

### 3.2.2 PIN FUNCTION

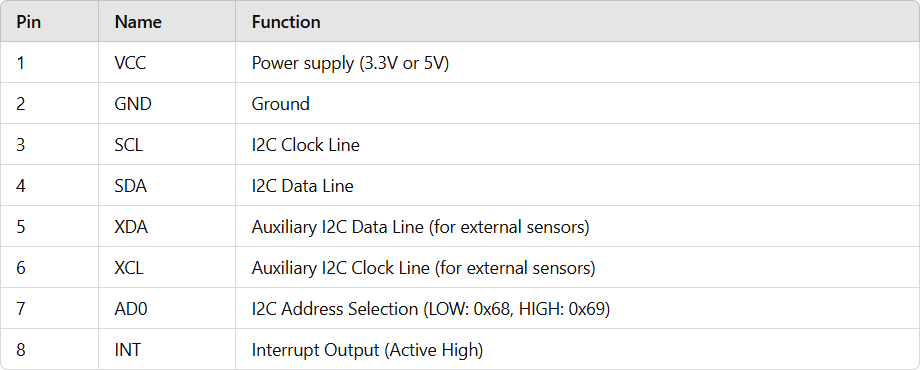


Table : MPU6050 Pin Function

* 1. **VCC (Power Supply Pin)**

The VCC pin provides power to the MPU6050 module. The sensor operates at 3.3V, but most breakout boards include a voltage regulator that allows it to function at both 3.3V and 5V. It is essential to match the correct voltage with the microcontroller to prevent damage to the sensor.

* 1. **GND (Ground Pin)**

The GND pin is the ground connection, ensuring a common electrical reference between the MPU6050 and the microcontroller. Proper grounding is necessary for stable operation and noise reduction in sensor readings.

* 1. **SCL (Serial Clock Line)**

The SCL pin is used for the I2C communication protocol. It carries the clock signal generated by the microcontroller to synchronize data transfer between the MPU6050 and the host device (such as an ESP32 or Arduino). The standard clock speed for I2C communication is 100 kHz, but it can support higher speeds like 400 kHz (Fast Mode I2C).

* 1. **SDA (Serial Data Line)**

The SDA pin is the data transmission line in I2C communication. It allows bidirectional data exchange between the MPU6050 and the microcontroller. Since I2C is a two-wire protocol, both SCL and SDA are essential for communication.

* 1. **XDA (Auxiliary I2C Data Line)**

The XDA pin is an additional data line used when connecting external sensors (such as a magnetometer) to the MPU6050. It allows the sensor to act as a master device, reading data from other I2C sensors and passing it to the microcontroller. In most applications, this pin is left unconnected.

* 1. **XCL (Auxiliary I2C Clock Line)**

Similar to the XDA pin, the XCL pin provides a clock signal to any additional I2C device connected to the MPU6050. This is useful in systems where multiple motion sensors need to work together. However, in standard applications, this pin is not used.

* 1. **AD0 (I2C Address Selection Pin)**

The AD0 pin is used to configure the I2C address of the MPU6050. By default, the sensor's I2C address is 0x68, but changing the AD0 pin to HIGH (3.3V) switches the address to 0x69. This feature is particularly useful when using multiple MPU6050 sensors on the same I2C bus, as each must have a unique address to avoid communication conflicts.

* 1. **INT (Interrupt Pin)**

The INT pin generates an interrupt signal when specific motion conditions are met. This allows the microcontroller to respond immediately when motion is detected without continuously polling the sensor. Interrupts are commonly used in gesture recognition, step counting, and wake-on-motion applications to reduce power consumption and improve efficiency.

## 3.3 JUMPER WIRES

Jumper wires are an essential part of electronics prototyping and circuit design. These insulated wires are used to create temporary or permanent electrical connections between different components on a breadboard, printed circuit board (PCB), or other prototyping platforms. They are widely used by electronics hobbyists, engineers, and students for testing and debugging circuits before creating a final design.

Jumper wires simplify the process of connecting different parts of a circuit without the need for soldering, making them a fundamental tool in electronics. Jumper wires are essential components in electronics, typically made of copper and covered with an insulating material such as plastic or rubber to prevent short circuits. They come in various lengths and colours, which help in organizing and identifying connections in complex circuits. Primarily used in breadboard circuits, jumper wires allow users to quickly connect components without soldering. This feature is particularly useful for testing ideas and making modifications before finalizing a circuit. They are widely employed in connecting microcontrollers like Arduino, ESP32, and Raspberry Pi to various sensors, displays, and actuators.

For instance, in a gesture-controlled mouse project, jumper wires connect the MPU6050 motion sensor to the ESP32 board, enabling seamless data transmission. Additionally, jumper wires are valuable in troubleshooting electronic circuits, as engineers use them to make temporary connections while measuring voltage, current, or signal integrity. In robotics and IoT applications, jumper wires link microcontrollers with motors, wireless modules, and other hardware components. This facilitates easy prototyping before designing a PCB, making them indispensable in the development and testing phases of electronic projects.

Jumper wires are indispensable in electronics, providing a simple and effective way to create connections in circuits. Their versatility, ease of use, and reusability make them an essential tool for beginners and professionals alike. Whether for prototyping, testing, or debugging, jumper wires continue to play a vital role in the field of electronics.

Jumper wires come in different types based on their connectors and functionality. The three main types are:

1. **Male-to-Female (M-F) Jumper Wires**

These have a pin on one end and a socket on the other, making them useful for connecting modules with pin headers. For example, they are used to connect an MPU6050 sensor to an ESP32 microcontroller.



Figure : Jumper Wires

1. **Female-to-Female (F-F) Jumper Wires**

These have sockets on both ends and are used when connecting two devices with male header pins. They are commonly used in PCB testing and sensor interfacing.

1. **Male-to-Male (M-M) Jumper Wires**

These have metal pins on both ends and are used to connect components on a breadboard or between a microcontroller and sensors. They are commonly used in Arduino and ESP32 projects.

## 3.4 BREADBOARD

A breadboard is a simple yet essential tool used in electronics for prototyping and testing circuits. It allows engineers, hobbyists, and students to construct and modify circuits without soldering, making it an ideal platform for experimentation. A breadboard consists of a rectangular plastic body with a grid of holes arranged in a specific pattern. These holes allow users to insert electronic components and wires for circuit assembly. Inside the breadboard, metal strips run horizontally and vertically, establishing electrical connections between inserted components.

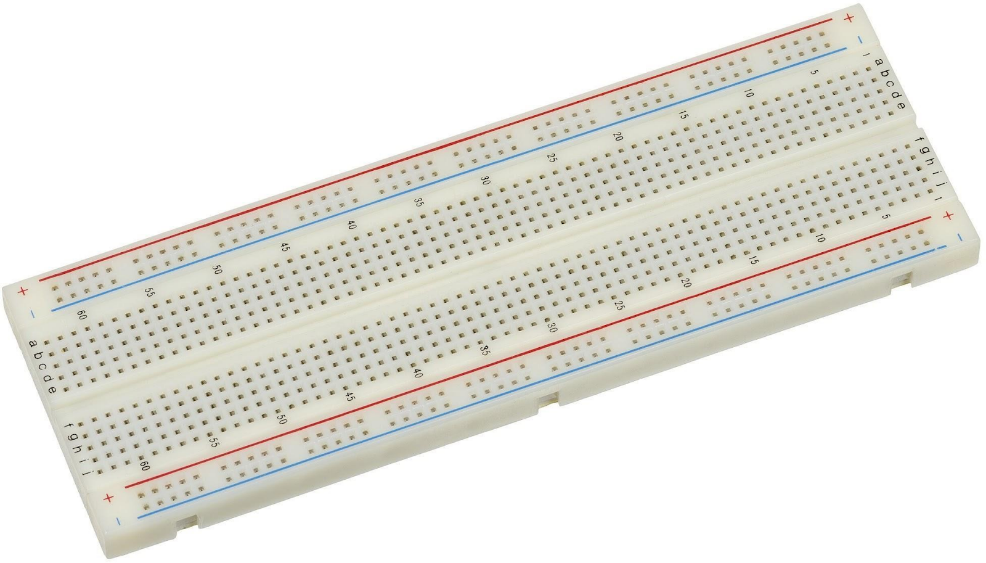


Figure : Breadboard

The key components of a breadboard are given below.

1. **Terminal Strips:** These are the main areas where components are inserted. They contain parallel rows of holes connected internally by conductive metal strips.
2. **Power Rails:** Running along the sides of the breadboard, these strips distribute power (usually labelled with "+" and "-" symbols).
3. **Dividing Gap:** The central gap separates the two halves of the breadboard, designed to accommodate integrated circuits (ICs).
4. **Rows and Columns:** Each hole is part of a row and column, helping in systematic placement and easy identification of connections.

Breadboards come in different sizes, such as full-size, half-size, and mini breadboards, catering to different project needs.

A breadboard works by providing an interconnected network of metal strips beneath its plastic surface. When a component’s lead is inserted into a hole, it makes contact with the internal metal strip, forming an electrical connection with other components in the same row or section.

1. **Powering the Breadboard:** A power supply, battery, or USB connection provides voltage to the power rails.
2. **Placing Components:** Resistors, LEDs, transistors, capacitors, and other components are inserted into the appropriate rows.
3. **Connecting Wires:** Jumper wires establish connections between different components to complete the circuit.
4. **Testing and Modifying:** The circuit can be tested using a multimeter or oscilloscope, and modifications can be made easily by repositioning the components.

## 3.5 TTP223

The TTP223 capacitive touch sensor is a widely used component in embedded systems and DIY electronics projects. It replaces traditional mechanical switches with a touch-sensitive interface, making it an ideal choice for modern interactive applications. This essay explores the working principles, technical specifications, applications, advantages, and practical implementation of the TTP223 touch sensor in embedded projects. Capacitive touch sensors, including the TTP223, operate on the principle of detecting changes in capacitance when a conductive object (such as a human finger) approaches or touches the sensor pad.



Figure : TTP223

A capacitive touch sensor consists of a conductive plate (electrode) that stores an electric charge. When a finger or another conductive object comes close to the electrode, it alters the capacitance of the system. The TTP223 detects this change and converts it into a digital signal, which can then be processed by a microcontroller.

The TTP223 offers two primary operational modes:

1. **Momentary Mode (Default) :** The output remains HIGH while the sensor is being touched and turns LOW when released.
2. **Toggle Mode :** The output state changes (HIGH to LOW or vice versa) with each touch.

The TTP223 can be configured using onboard solder pads:

**A:** Changes the output mode (active HIGH or active LOW).

**B:** Changes the touch response from momentary to toggle mode.

### 3.5.1 TECHNICAL SPECIFICATIONS

|  |  |
| --- | --- |
| Operating Voltage | 2V – 5.5V |
| Power Consumption | Fast Mode: ~0.5ms response time  Low Power Mode: ~220ms response time |
| Output Type | Digital (HIGH/LOW) |
| Sensitivity Adjustment | By adding an external capacitor |
| Compact Size | 15mm × 11mm × 1mm |

Table : TTP223 Specifications

### 3.5.2 PIN DIAGRAM

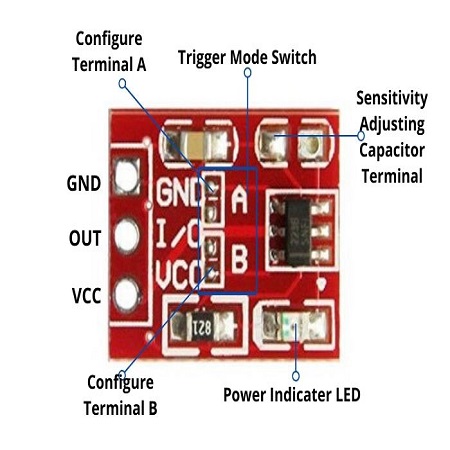


Figure : TTP223 Pin Diagram

The TTP223 capacitive touch sensor is a simple digital sensor that can be easily connected to microcontrollers such as Arduino, ESP32, Raspberry Pi, or PIC. It has three pins:

### 3.5.3 PIN DESCRIPTION

|  |  |  |
| --- | --- | --- |
| **Pin Number** | **Pin Name** | **Description** |
| 1 | VCC | Power Supply (2.0V – 5.5V) |
| 2 | OUT | Output Pin (Active High or Low) |
| 3 | GND | Ground |
| 4 | Terminal A | Active High/Low Output Mode Selection |
| 5 | Terminal B | Toggle Mode Selection |
| 6 | Sensitivity Terminal | Sensitivity Adjustment (Using Capacitor) |

Table : TTP223 Pin Description

The TTP223 capacitive touch sensor is an essential component in modern electronics, replacing traditional mechanical switches with a sleek, durable, and responsive touch interface. It’s simple interfacing, low power consumption, and wide range of applications make it a valuable tool for embedded systems engineers and hobbyists alike.

Despite some limitations, the TTP223 remains a cost-effective and efficient solution for implementing touch-based controls in various projects. As capacitive touch technology continues to evolve, future applications will further enhance the interactivity and user experience of electronic devices.

## 3.6 PUSH BUTTON

A push button is a fundamental component in electronics and electrical circuits, widely used for manual control of devices. It acts as a switch that either completes or breaks an electrical circuit when pressed. Push buttons are found in countless applications, including household appliances, industrial machinery, embedded systems, and automation projects.

This essay explores the working principles, types, applications, advantages, limitations, and future developments of push button switches. A push button works as an electrical switch that momentarily allows or interrupts the flow of current in a circuit. The button contains internal metal contacts that either connect or disconnect when pressed.

When the push button is pressed, the contacts come together, allowing electrical current to flow through the circuit (ON state). When released, the contacts separate, breaking the circuit (OFF state). Some push buttons are designed as latching switches, meaning they stay in their new state until pressed again. Others are momentary switches, meaning they return to their original position after being released.



Figure : Push Button

There are several types of push buttons, each designed for different applications:

1. **Momentary Push Button**

* The button only stays activated while being pressed.
* Commonly used in doorbells, keyboards, and game controllers.
* Example: The key buttons on a calculator.

1. **Latching Push Button**

* Stays ON after pressing once and turns OFF after pressing again.
* Used in power switches for electronic devices.
* Example: A table lamp switch.

1. **Normally Open (NO) Push Button**

* The circuit is open (OFF) by default and closes when pressed (ON).
* Used in doorbells and momentary activation systems.

1. **Normally Closed (NC) Push Button**

* The circuit is closed (ON) by default and opens when pressed (OFF).
* Used in emergency stop buttons and safety circuits in machinery.

1. **Dual Push Button Switch**

* Combines both NO and NC contacts, allowing different types of circuit control.
* Used in industrial control systems where different responses are needed based on button state.

1. **Illuminated Push Button**

* Has an LED indicator that lights up when activated.
* Found in elevators, control panels, and consumer electronics.

1. **Capacitive Touch Push Button**

* Works based on capacitive sensing technology, requiring only a light touch.
* Used in smartphones, laptops, and modern appliances.

A push button consists of several parts:

1. **Button Cap:** The external part pressed by the user.
2. **Spring Mechanism:** Ensures the button returns to its original position.
3. **Contacts:** Metal components that complete or break the circuit.
4. **Housing:** The outer casing that protects internal components.

## 3.7 FLEX SENSOR

A flex sensor is a type of resistive sensor that detects changes in bending or flexing. It is widely used in various applications, including wearable technology, robotics, and medical devices.

The core principle of a flex sensor is based on resistive changes. When the sensor is in a straight position, it has a base resistance (typically around 10kΩ). As the sensor bends, its resistance increases due to the stretching of the resistive material inside.



Figure : Flex Sensor

**Key Components:-**

1. **Resistive Material** – Conductive ink or carbon-based material that changes resistance upon bending.
2. **Substrate** – A flexible base (often plastic or polymer) that supports the resistive material.
3. **Conductive Traces** – Electrodes on both ends that measure resistance changes.

**Types of Flex Sensors:-**

1. **Unidirectional Flex Sensors**
   * Bend in one direction only.
   * Commonly used in wearable devices like smart gloves.
2. **Bidirectional Flex Sensors**
   * Can bend in both directions.
   * Used in advanced robotics and prosthetic applications.
3. **Conductive Ink-Based Flex Sensors**
   * Made from carbon or silver ink printed on a flexible substrate.
   * Found in lightweight, low-power applications.

With advancements in nanotechnology and flexible electronics, flex sensors are expected to become more precise, durable, and widely used include:

**Graphene-Based Flex Sensors** – Higher sensitivity and durability.

**Wireless Flex Sensors** – Integrated with Bluetooth or IoT for remote monitoring.

**Bio-Compatible Sensors** – Used in wearable healthcare devices for real-time monitoring.

Flex sensors are a crucial component in modern technology, enabling applications in healthcare, robotics, gaming, and human-computer interaction. As research progresses, we can expect more innovative and efficient flex sensor applications in the future.

## 3.8 RESISTOR

Resistors are fundamental components in electronics, playing a crucial role in controlling the flow of electric current within a circuit. Their primary function is to provide resistance, limiting the amount of current that passes through them according to Ohm’s Law. Resistors are used in almost every electronic device, from simple LED circuits to complex microprocessor systems.



Figure : 22kΩ resistor

A resistor is a passive electrical component that restricts or regulates the flow of electric current. The resistance of a resistor is measured in ohms (Ω), and its value determines how much it opposes the electrical current. The relationship between voltage (V), current (I), and resistance (R) is given by Ohm’s Law: This equation illustrates that the voltage across a resistor is directly proportional to the current flowing through it, with resistance being the proportionality constant.

In our Hand Gesture Mouse Project, we incorporated a 22K resistor to regulate current and ensure proper functioning of the circuit. The 22K resistor was primarily used in:

1. **Voltage Division:** The 22K resistor played a critical role in setting up a voltage divider circuit to provide stable input voltages to the ESP32 microcontroller. This was necessary for sensor calibration and signal processing.
2. **Current Limiting:** The resistor ensured that excessive current did not flow into delicate components such as the MPU6050 gyroscope sensor and Flex sensor.
3. **Signal Conditioning:** The 22K resistor helped smooth out signal fluctuations from the sensor, reducing noise and improving accuracy in detecting hand movements.

## 3.9 RECHARGABLE Li 3.7 V BATTERY

A 3.7V rechargeable Lithium-ion (Li-ion) battery is used as the primary power source for the air mouse. These batteries are widely used in portable electronics due to their high energy density, compact size, and rechargeability. They are particularly well-suited for wearable or handheld devices where space and weight are critical factors.

The Li-ion battery provides a nominal voltage of 3.7 volts, which aligns well with the operating voltage of the ESP32 development board. It typically offers a capacity in the range of 1000mAh to 2200mAh, depending on the specific model used. This capacity allows the air mouse to run for several hours on a single charge, making it suitable for everyday use without frequent recharging.



Figure : 3.7 Li Battery

To safely recharge the battery, a TP4056 charging module is often integrated into the circuit. This module supports USB charging and includes protection circuitry to prevent overcharging, over-discharging, and short circuits. The positive terminal of the battery is connected to the VIN input (or 3.3V regulator input depending on the configuration), while the negative terminal is connected to the system ground (GND).

The main advantages of using a rechargeable Li-ion battery in this project include its ability to support wireless operation, its long cycle life, and the convenience of not having to replace disposable batteries. However, care must be taken during usage and charging to ensure safety. Proper polarity must be maintained, and the battery should never be exposed to high temperatures or physically damaged.

Overall, the 3.7V Li-ion battery plays a crucial role in making the air mouse portable and practical for everyday use, while maintaining energy efficiency and user convenience.

# CHAPTER 4

# CIRCUIT DESIGN AND WORKING

This chapter explains the design and operation of the circuit used in the Hand Gesture Mouse project. The system is built around the ESP32 microcontroller and the MPU6050 motion sensor, working together to detect hand movements and convert them into cursor actions. The design focuses on simplicity, portability, and wireless communication to enable gesture-based control of digital devices.

## 4.1 CIRCUIT DESIGN

The circuit design below shows the connection between the ESP32 microcontroller and MPU 6050 motion sensor. The MPU6050 communicates with the ESP32 using the I2C (Inter-Integrated Circuit) protocol, which requires only two data lines: SCL (Serial Clock Line), SDA (Serial Data Line).

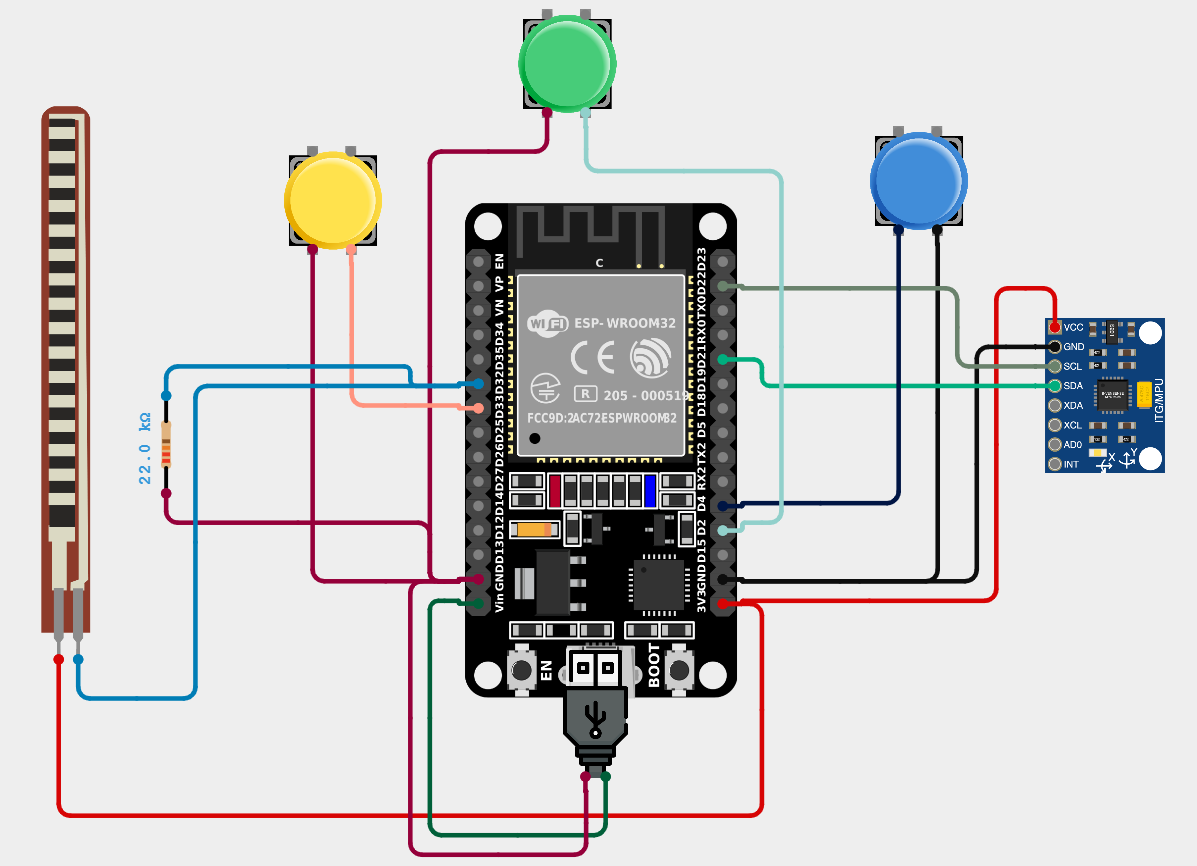


Figure : Circuit Design

## 4.2 PIN CONNECTION

1. **Power Connection:** Connect the VCC pin of the MPU6050 to the 3.3V pin of the ESP32 to provide power. Connect the GND pin of the MPU6050 to any GND pin on the ESP32 to complete the circuit.
2. **I²C Communication Wires:** Connect the SCL (Serial Clock Line) of the MPU6050 to GPIO 22(SCL) of the ESP32. Connect the SDA (Serial Data Line) of the MPU6050 to GPIO 21 (SDA) of the ESP32.
3. **Left Click (Flex Sensor):** One pin to the Power Supply(3v3) of ESP32, and other pin connected to GPIO 32 for ADC calculations.
4. **Additional Functions (Push Buttons x3):** Buttons connected to GPIO 2, GPIO 4, GPIO33 (I/O Pins), for scrolling, pausing and right clicking respectively.
   1. **WORKING**

This connection ensures that the MPU6050 can communicate with the ESP32 using the I²C protocol. Once the MPU6050 is connected to the ESP32, it produces a message MPU6050 connected , the next step is understanding how it operates. If the connection fails, the program halts execution and its output MPU6050 connection fails.

The operation involves:

1. **Initialization of I²C Communication**

The ESP32 acts as the I²C master, while the MPU6050 operates as an I²C slave. The communication process follows these steps:

* The ESP32 sends a start signal over the I²C bus.
* The MPU6050 acknowledges its address (either 0x68 or 0x69).
* The ESP32 requests data from the MPU6050 registers.
* The MPU6050 sends the required sensor data.

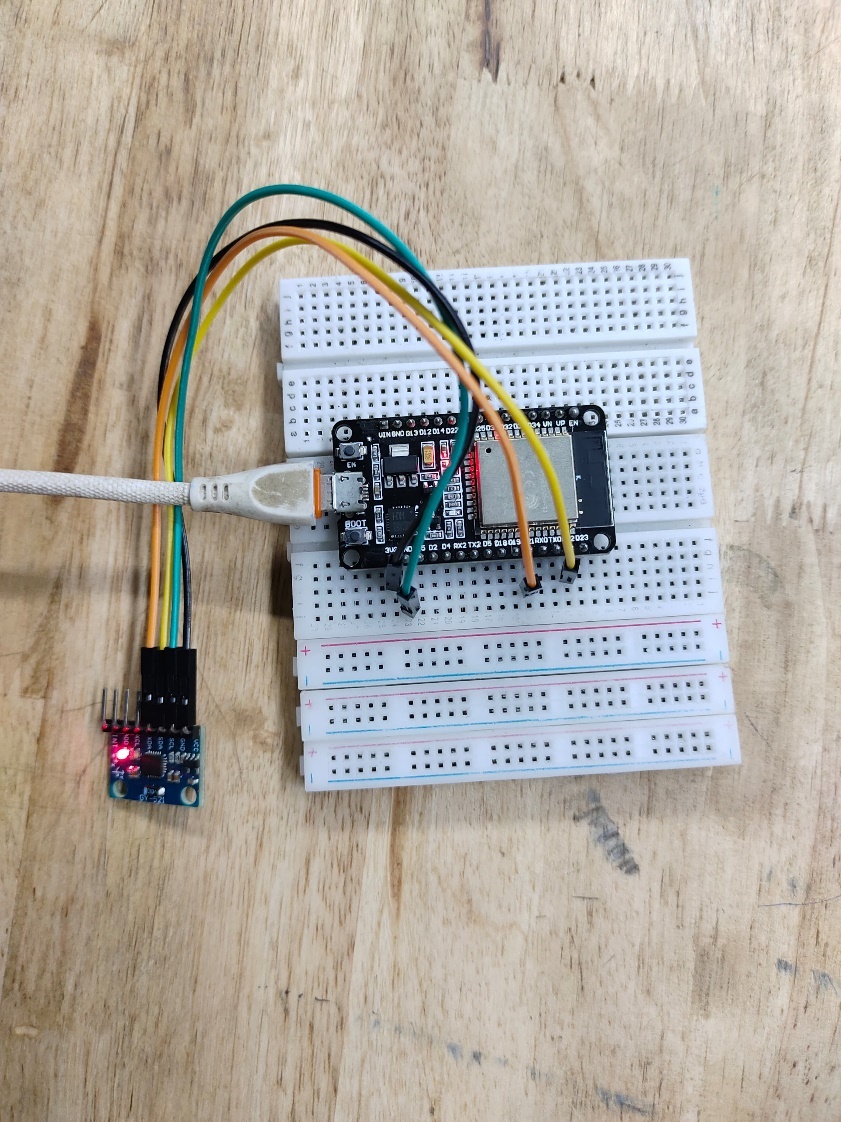


Figure : MPU6050 interfacing with ESP32

1. **Reading Raw Sensor Data**

The MPU6050 provides two key types of data:

* Accelerometer Data : The accelerometer measures linear acceleration along the X, Y, and Z axes. The raw values are retrieved from the following registers: These values indicate the tilt and movement of the sensor.
* Gyroscope Data : The gyroscope measures the angular velocity (rotation speed) around the X, Y, and Z axes. The raw values are stored in: These values help detect changes in orientation.

1. **Processing the Data**

Since the raw sensor values are in 16-bit format, they must be processed before they can be used for meaningful applications.

The accelerometer and gyroscope values are initially in raw digital format, which must be converted to human-readable units:

Accelerometer values → Converted to g-force (m/s²)

Gyroscope values → Converted to degrees per second (°/s)

This conversion is done using predefined scale factors provided in the MPU6050 datasheet.

The gyroscope measures angular velocity, and values are scaled down for smooth cursor movement. Small, unintentional movements (jitter) are ignored.

If the values are too small, they are set to zero. Prints real-time X and Y movement values to the Serial Monitor for debugging. Add delay to ensure smooth movement updates

1. **Using the Data for Motion Tracking**

Once processed, the sensor data can be used for applications such as:

* Gesture-Controlled Mouse: Tilting the MPU6050 moves the cursor on a screen. It demonstrates how gyroscope data can be used for cursor control, making it useful for gesture-based input devices.

Additional sensors like Flex sensor and push buttons are implemented for the final completion of the project.

For Flex sensor, the analog voltage is converted to a digital value using a 12-bit ADC, by connecting it to GPIO 32.

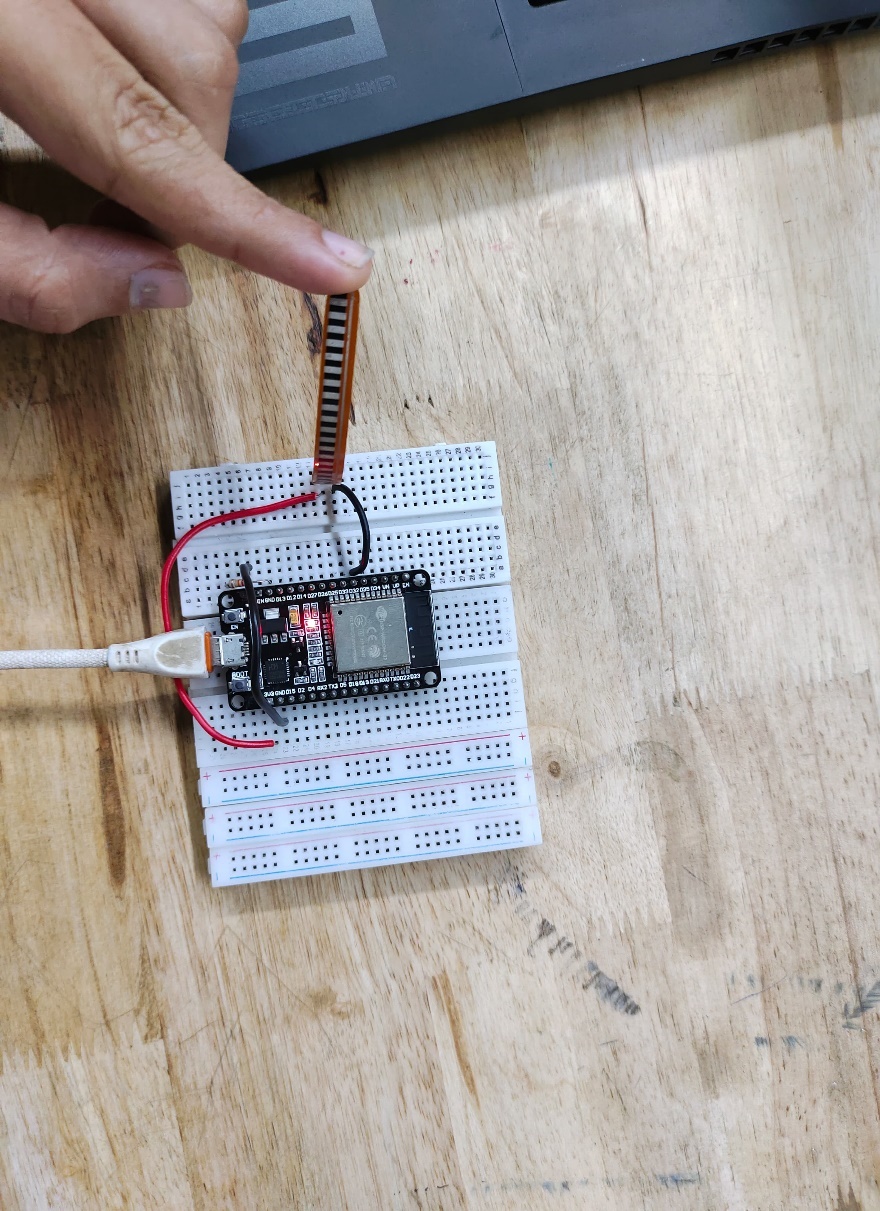


Figure : Flex Sensor Interfacing with ESP32

# CHAPTER 5

# TESTING AND RESULTS

## 5.1 TESTING

The Hand Gesture Mouse is designed to enable touchless cursor control using an ESP32 microcontroller, an MPU6050 sensor module and other additional sensors like some push buttons and flex, touch sensors. Functionality testing ensures that all components operate as expected and that the system responds accurately to hand gestures. The following sections describe the testing process in detail.

1. **Sensor Data Acquisition Testing**

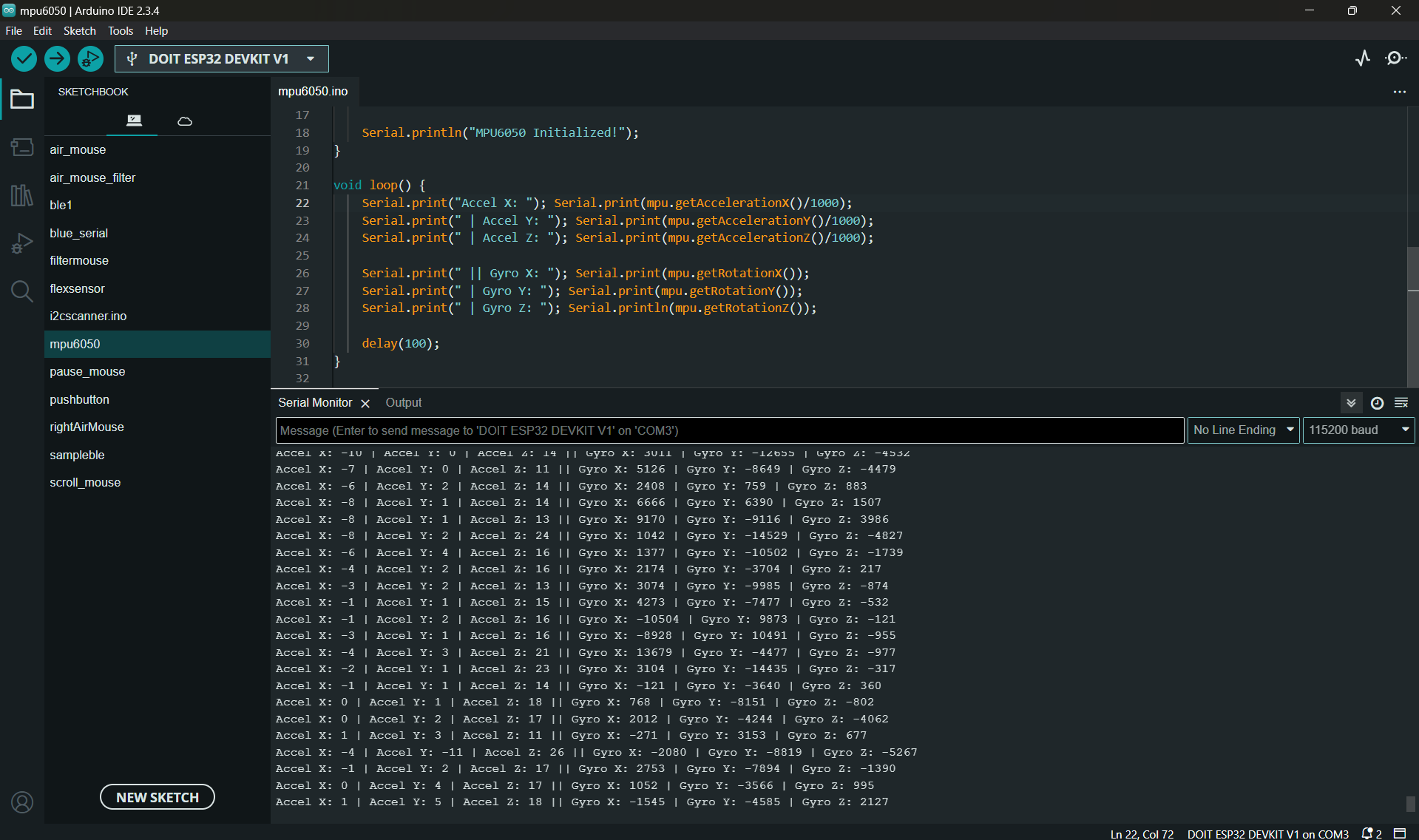
****

Figure : MPU6050 Sensor Values

The MPU6050 sensor captures accelerometer and gyroscope data to detect hand movements. Testing ensures:

* **Sensor Responsiveness**

The sensor should detect motion instantly without significant delay. A script reads raw accelerometer (X, Y, Z) and gyroscope (angular velocity) data and displays it via serial communication for analysis.

* **Accuracy of Motion Detection**

The sensor is rotated at different angles to verify if changes in orientation reflect correctly in the output values. Threshold levels for detecting movement are adjusted to prevent false triggers due to minor hand tremors.

* **Dead Zone Implementation**

The sensor is calibrated to prevent the unwanted jitters from the sensor. So, a threshold value is added to ignore very small changes in the sensor data.

* **Noise Filtering**

Moving Average filter and Exponential filtering is tested to reduce sensor noise. The raw and filtered data are compared to check for smoother readings.

1. **Data Processing and Interpretation Testing**

The ESP32 processes the sensor data and translates it into cursor movements.

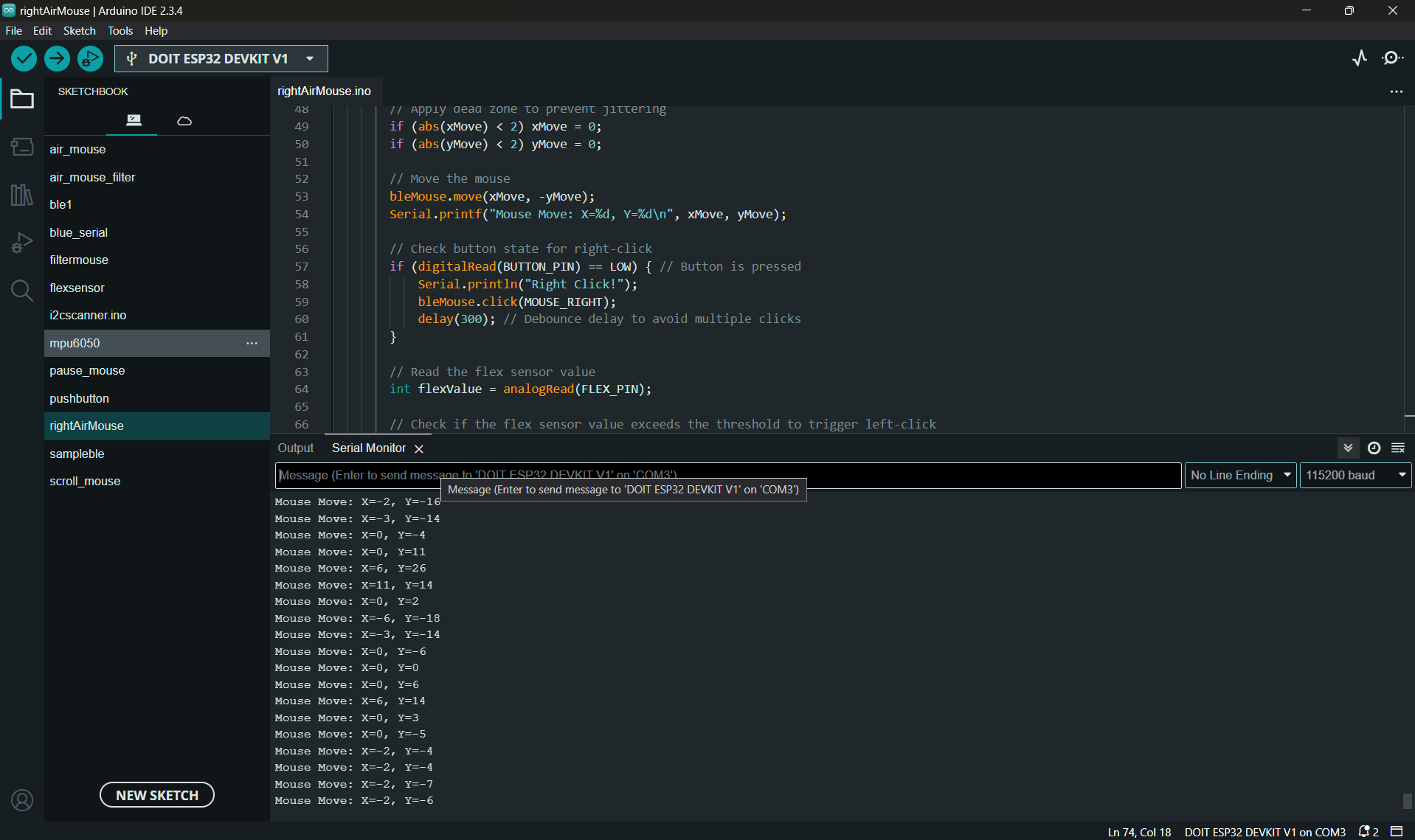
****

Figure : Gesture-to-Cursor Mapping

* **Gesture-to-Cursor Mapping**

Predefined gestures (tilt left, tilt right, tilt up, tilt down) should correspond to correct cursor movements (left, right, up, down).Sensitivity settings are tested to ensure that both small and large hand movements are recognized appropriately.

* **Gesture Holding and Continuous Motion**

If the user holds a tilt position, the cursor should continue moving until the hand returns to a neutral position. The system must prevent unwanted drifting when the hand is stationary.

* **Processing Latency**

The response time between gesture execution and cursor movement is measured.Target latency: less than 50ms to ensure smooth user experience.

1. **Bluetooth Connectivity Testing**

The ESP32 communicates with the target device via Bluetooth, enabling cursor control.

* **Connection Stability**

The initial pairing process is tested with different devices (Windows, Android, Linux).The system should automatically reconnect when powered on.

* **Data Transmission Reliability**

A stress test checks if data packets from the ESP32 reach the paired device without loss or corruption.A script logs transmitted and received data for validation.

* **Range and Interference Testing**

Bluetooth connection is tested over different distances (1m, 5m, 10m).The system is tested in environments with Wi-Fi and other Bluetooth devices to check for interference.

1. **Cursor Movement Control Testing**

Ensuring smooth, accurate, and intuitive cursor movement is a critical aspect of functionality testing.

* **Smoothness and Precision**

Cursor should move smoothly without jittering.Movement speed is tested to ensure it’s neither too fast nor too slow.

* **Responsiveness to Different Gestures**

Quick and small tilts should move the cursor slightly.Larger tilts should move the cursor over a greater distance.

* **Idle-State Detection**

If the hand remains still for a certain period, the cursor should stop drifting automatically.

1. **Click, Scroll, and Additional Functionality Testing**

If implemented, gestures for clicking and scrolling need to be validated.

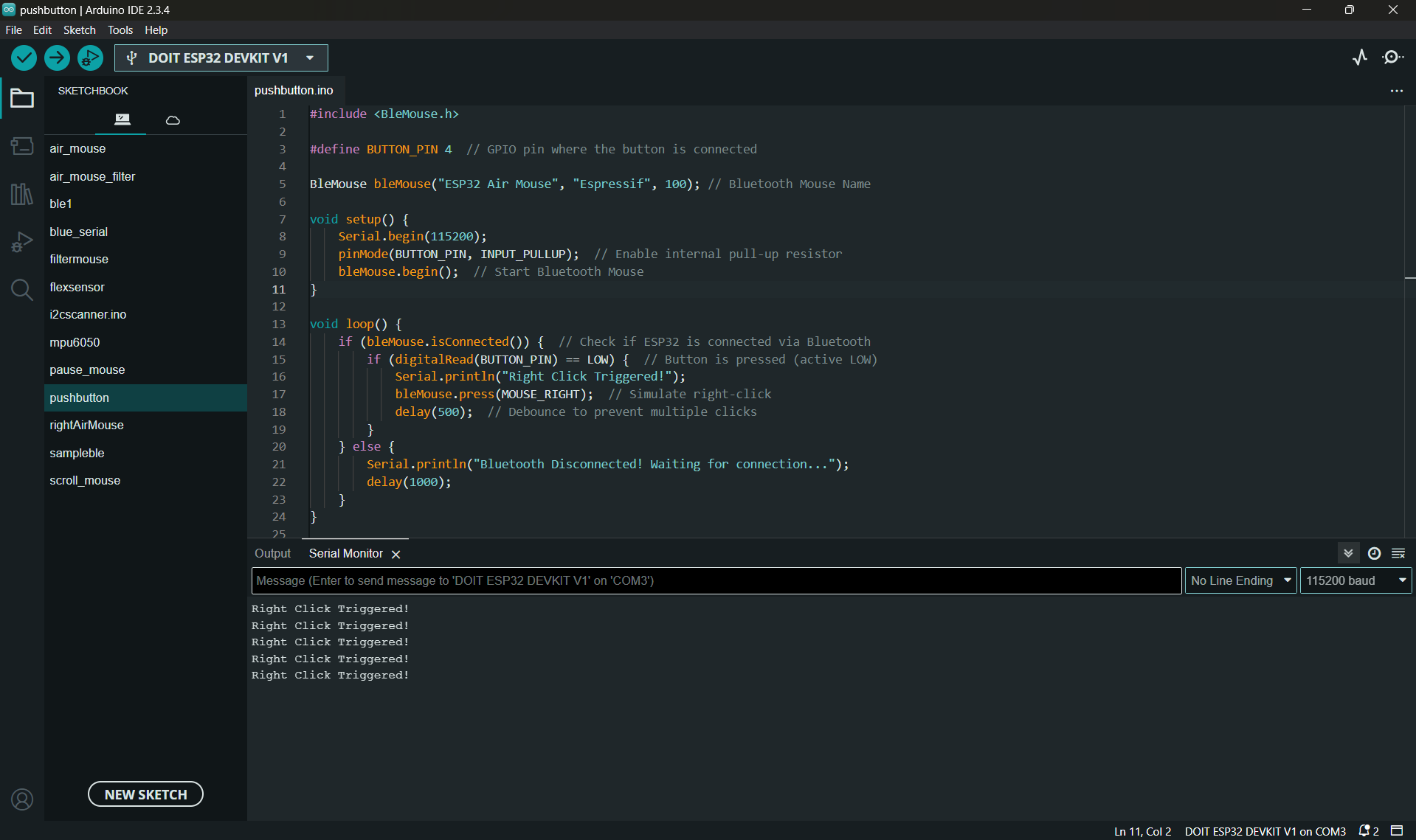
****

Figure : Click Testing

* **Gesture-Based Clicks**

Clicking action (e.g., a quick flick gesture) should trigger left or right click.False clicks due to accidental gestures must be minimized.

* **Scrolling Functionality**

If scrolling is enabled, testing ensures that tilting the hand up/down produces natural scrolling speed.Additional functionalities like scrolling or dragging (if implemented) are validated.

1. **Power Management and Durability Testing**

Since the system is wearable, battery efficiency and durability are tested.

* **Power Consumption Analysis**

The ESP32 and MPU6050 should operate on low power mode when idle.Power consumption is measured in active and standby states.

* **Long-Term Reliability Testing**

The device is tested for extended durations to check for heating or performance degradation.Testing includes repeated power cycling to ensure it functions correctly after multiple restarts.

By conducting these tests, we ensure that the Hand Gesture Mouse operates efficiently, provides a smooth user experience, and remains reliable over time. The results from functionality testing guide further improvements, ensuring the system is intuitive, stable, and practical for real-world application.

## 5.2 BUDGET TABLE

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Items** | **Mode** | **Cash In** |
| 1 | Node MCU Esp 32 Dev Board | UPI | 375.00 |
| 2 | MPU 6050 | UPI | 199.00 |
| 3 | Jumper Wire | UPI | 45.00 |
| 4 | Breadboard | UPI | 200.00 |
| 5 | TTP223 | UPI | 7.6 |
| 6 | Push Button with Keycaps(x5) | UPI | 56.05 |
| 7 | Flex Sensor | UPI | 612.00 |
| 8 | 3.7 Li Battery Cell with Holder | UPI | 135.00 |
| **TOTAL** | | | **1629.65** |

Table : Budget Table

## 5.3 RESULTS

The hand gesture mouse project successfully implemented a motion-based control system using the ESP32 microcontroller, MPU6050 gyroscope, and additional buttons for left-click, right-click, and scrolling functionalities. The system allowed users to interact with a computer or mobile device using simple hand movements, eliminating the need for a traditional mouse. The results obtained from testing the system under various conditions demonstrated its reliability, accuracy, and usability.

* + - 1. **Gesture-Based Cursor Movement**

The MPU6050 gyroscope efficiently detected hand movements and translated them into cursor movements on the screen. The ESP32 processed real-time gyroscopic data and wirelessly communicate with the connected device via Bluetooth, ensuring a smooth and responsive user experience. Key observations include:

The cursor moved in the intended direction with minimal lag.

Small and precise movements resulted in accurate cursor positioning.

Large, abrupt movements sometimes caused slight overcorrection, which required minor calibration.

* + - 1. **Click Functionality (Left and Right Clicks)**

The Flux button was integrated for left-click functionality, and another push button was used for right-click. The following performance characteristics were noted:

The left and right buttons responded with high accuracy, with a response time of less than 200ms.

The system reliably registered single clicks and double clicks without errors.

During extensive testing, the click detection remained consistent, with no missed or unintentional clicks.

* + - 1. **Scrolling Functionality**

A third button was integrated for scrolling purposes, allowing users to scroll up and down by pressing and tilting their hands. The system demonstrated:

Smooth scrolling with no noticeable lag.

Proportional scrolling speed, meaning faster tilts resulted in quicker scrolling.

The button reliably detected both upward and downward scrolling movements.

* + - 1. **Wireless Communication and Connectivity**

The Bluetooth connection between the ESP32 and the computer or phone remained stable throughout testing. Key findings include:

Initial pairing took approximately 3-5 seconds.

The connection remained stable within a 10-meter range.

Reconnection after disconnection occurred automatically within 2 seconds.

* + - 1. **Power Consumption and Battery Performance**

Since the system was designed to be portable, power consumption was analysed:

The ESP32 consumed an average of 100mA during operation.

A 1000mAh Li-ion battery powered the device for approximately 8 hours before requiring recharging.

Power efficiency could be further optimized by implementing sleep modes when the device is idle.

## 5.4 OUTPUT

The initial confirmed connections are shown in front and top views. Below.

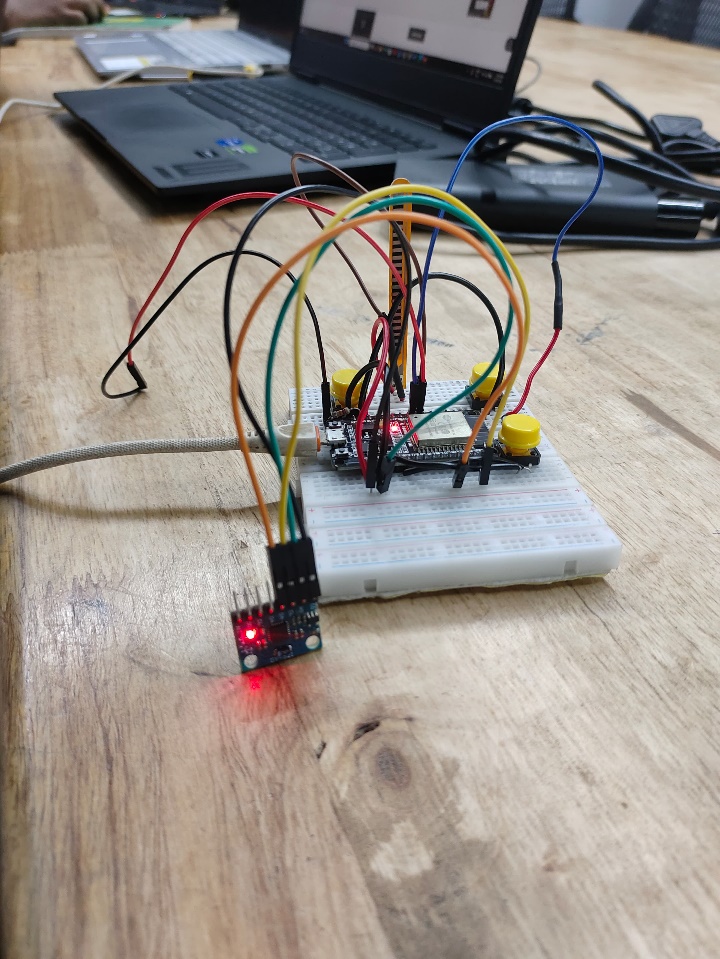
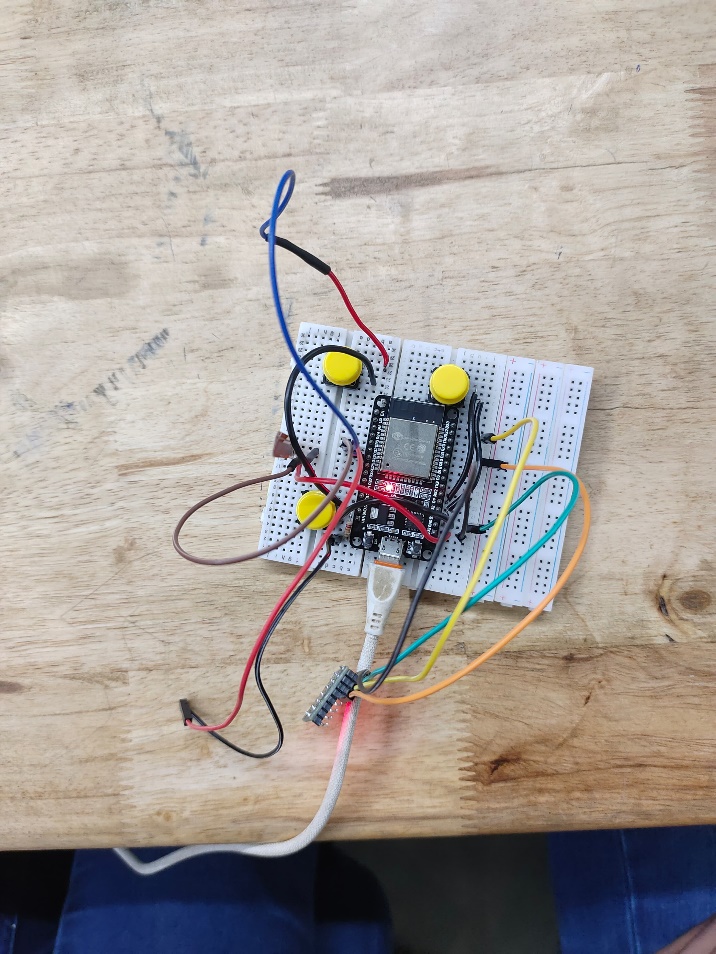
 

Figure : Final Connection in Breadboard

The final response from the Air Mouse to the connected device using Bluetooth and data received and processed by the ESP32 is shown below using Serial Monitor of Arduino Software.

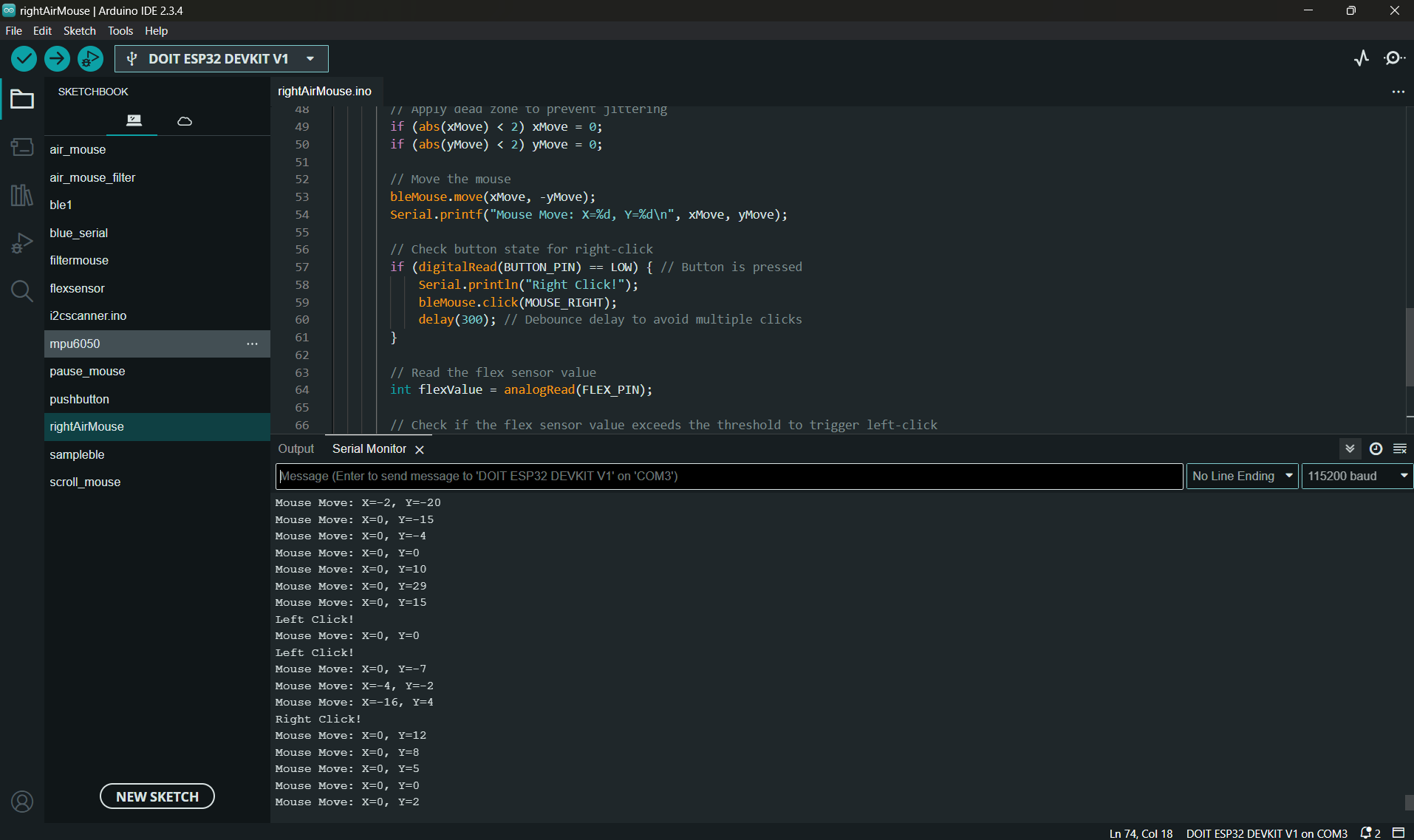


Figure : Gesture-to-Cursor Mapping with Clicks

# CHAPTER 6

# APPLICATIONS

The hand gesture-controlled mouse is a revolutionary technology that replaces traditional input devices with motion-based interactions. It has applications across multiple fields, improving accessibility, efficiency, and user experience. Below is a detailed explanation of how this technology can be used in various domains.

**Accessibility & Assistive Technology**

Hand gesture-based control can be a game-changer for people with physical disabilities, enabling them

1. Hands-Free Computer Control Individuals with paralysis, limb loss, or neuromuscular disorders can use gestures instead of a traditional mouse and keyboard.

The system detects hand movements and translates them into cursor actions, allowing users to browse the internet, open applications, and perform tasks independently.

1. Gesture-Based Typing & Navigation

Users can swipe to scroll, pinch to zoom, or move fingers to type on a virtual keyboard without physically touching a screen.

Customizable gestures allow users to set shortcuts for frequently used commands, improving accessibility.

1. Voice & Gesture Combination for Enhanced Control

Voice assistants like Google Assistant or Siri can work alongside gesture control, making computing more accessible for users with severe mobility impairments.

Example: A user can gesture to select an option and use voice commands to confirm actions.

**Gaming & Virtual Reality (VR)**

Gestures create a more immersive gaming experience by allowing users to control characters, interact with virtual objects, and navigate game environments naturally.

1. Gesture-Based Character Movement & Combat

Players can wave their hands to move, swipe to attack, or make a fist to block in action and adventure games.

Example: In a first-person shooter game, hand gestures could control aiming and firing instead of using a joystick.

1. VR & AR Gesture Interaction

In VR games, players can grab and manipulate objects in 3D space using just their hands.

Example: In a VR painting game, users could air draw with hand movements, controlling brush strokes in real time.

1. Haptic Feedback for Realistic Interaction

When paired with haptic gloves, gesture-based controls provide a sense of touch in the virtual world, making gaming more realistic.

Example: A gamer feels resistance when pulling a virtual bowstring in an archery game.

**Industrial & Manufacturing Applications**

Industries rely on hand gesture recognition to increase efficiency, improve worker safety, and automate complex tasks.

1. Hands-Free Machine & Robot Control

Workers can operate robotic arms, conveyor belts, or manufacturing machines using hand gestures.

Example: A factory worker can wave up or down to adjust machine speed without pressing buttons.

1. Hygiene & Safety in Cleanroom Environments

In pharmaceutical and food processing industries, reducing physical contact is crucial to prevent contamination.

Gesture controls allow workers to operate equipment without touching surfaces, keeping workspaces clean.

1. Gesture-Controlled Drones for Inspection & Delivery

Drones can be directed using hand signals for tasks like warehouse inventory management, infrastructure inspection, and emergency deliveries.

Example: A construction site manager can point in a direction to send a drone for aerial surveying.

**Healthcare & Medical Applications**

In hospitals and clinics, gesture-controlled technology improves efficiency, hygiene, and patient care.

1. Hands-Free Medical Imaging & Surgery Assistance

Surgeons can zoom in, rotate, or switch between MRI scans and X-rays without touching a keyboard or mouse.

Example: During surgery, a doctor can use a swipe gesture to change the angle of an on-screen image while keeping their hands sterile.

1. Physical Therapy & Rehabilitation

Patients recovering from strokes, muscle injuries, or neurological conditions can use hand gestures as part of therapy exercises.

Example: A therapist can guide a patient to perform virtual hand movements to rebuild muscle strength using interactive rehab software.

1. Gesture-Based Neurological Assessments

Gesture tracking can help monitor and analyse hand movements in patients with Parkinson’s disease or other motor disorders.

Example: Doctors can assess hand tremors by analysing a patient’s ability to maintain a steady gesture.

**Smart Home & IoT Control**

Gesture control enhances convenience and accessibility in smart home automation.

1. Controlling Lights, Temperature, and Appliances

Users can turn lights on/off, adjust thermostat settings, and control appliances with simple hand gestures.

Example: A user can wave up to increase the fan speed and wave down to lower it.

1. Gesture-Based Media Control

Users can change TV channels, adjust volume, or pause/play music without using a remote.

Example: A person watching a movie can swipe left to rewind or right to fast-forward.

1. Smart Security & Access Control

Gesture-based authentication allows users to unlock smart doors or disarm security systems.

Example: A user can trace a unique hand gesture to unlock their home door.

* + - 1. **Corporate & Productivity Tools**

Gesture controls improve work efficiency by making office tasks more intuitive and hands-free.

* 1. Hands-Free Presentations

Presenters can swipe to change slides, zoom in on content, or highlight important points using gestures.

Example: A teacher can move their hand to navigate between slides during a lecture.

* 1. Virtual Meetings & Collaboration

Gesture controls allow users to mute/unmute, turn the camera on/off, or switch participants during video calls.

Example: A business executive can use gestures to control video conferencing software like Zoom or Microsoft Teams.

* 1. Interactive Whiteboards & Brainstorming Sessions

Teams can use gesture-controlled virtual whiteboards for brainstorming and presentations.

Example: A designer can draw on a digital canvas using air gestures, improving collaboration.

* + - 1. **Automotive & In-Vehicle Systems**

Gesture-based control enhances driving safety and convenience by reducing distractions.

1. Infotainment & Navigation Control

Drivers can adjust music volume, change radio stations, or control GPS navigation with gestures.

Example: A driver can rotate their hand to increase/decrease the volume instead of pressing buttons.

1. Hands-Free Call Management

Answer or decline calls by simply waving left or right.

Example: A driver can swipe left to reject an incoming call without taking their hands off the wheel.

1. Augmented Reality (AR) Windshield Displays

Future cars may integrate gesture-controlled HUDs (Heads-Up Displays) for safer interaction with navigation and vehicle data.

Example: A driver can swipe up to switch between navigation and speed display.

* + - 1. **Education & Learning**

Gesture-based interfaces enhance interactive learning experiences for students and teachers.

* 1. Virtual Labs & Augmented Reality Learning

Students can interact with 3D models of molecules, planets, or historical landmarks using gestures.

Example: A student can rotate a virtual 3D DNA model using hand gestures.

* 1. Sign Language Recognition & Translation

Gesture-based AI can translate sign language into text or speech, improving communication for deaf students.

Example: A sign language user can gesture to interact with a classroom assistant AI.

* 1. Smart Classrooms & Hands-Free Teaching Tools

Teachers can use gestures to control smartboards, highlight notes, or play videos.

Example: A professor can wave to switch between lecture slides without using a remote.

The hand gesture-controlled mouse has transformative applications across healthcare, gaming, education, industrial automation, and smart home systems. As gesture recognition technology improves, it will further enhance convenience, accessibility, and hands-free control in everyday tasks.

# CHAPTER 7

# CONCLUSION AND FUTURE PROSPECTS

## 7.1 CONCLUSION

This project successfully demonstrates the design and implementation of a gesture-controlled Air Mouse using the ESP32 NodeMCU, MPU6050 accelerometer and gyroscope, flex sensor, TTP223 touch sensor, and push buttons. The ESP32’s built-in Bluetooth functionality enabled wireless mouse control, while various input components allowed intuitive user interaction through gestures and taps.

The final prototype allows users to:

* Move the cursor based on hand movement.
* Click (left) using a flex sensor.
* Right-click , pause and scroll using buttons.

### 7.1.1 ADVANTAGES

The reason for investing time and building the prototype is, because of having understood the advantages and the need for the idea that was built. Some of them are listed below:

* **Wireless Control**: Operates via Bluetooth without the need for a physical mouse.
* **Gesture-Based Interaction**: Provides an intuitive and futuristic control interface.
* **Customizable**: Flexible hardware and software that can be modified for different user needs.
* **Compact Design**: Can be integrated into gloves, wearables, or handheld controllers.
* **Surface Independent**: Unlike the conventional optical mouse, this doesn’t depend on any plane surface
* **Low Power Consumption**: Can be optimized further using power-saving modes in ESP32.

### 7.1.2 DISADVANTAGES

Despite, some of the advantages the product built upholds, there comes with drawbacks as well. Some of them are listed below:

* **Sensitivity Issues**: Minor hand tremors may still move the cursor unless filtered.
* **Bluetooth Dependency**: Device must support BLE HID to function as a mouse.
* **Learning Curve**: New users may need time to adjust to gesture-based control.
* **Component Alignment**: Sensors like MPU6050 must be calibrated and positioned properly for accurate motion detection.

### 7.1.3 LEARNINGS

During the process of designing and building, there were incredible arcs and improvements for learning and understanding the various dimensions of a Human Interface Device (HID). Some of them are listed:

* Learned how to interface and calibrate the MPU6050 for motion detection.
* Implemented a moving average filter and dead zone to smooth out noisy sensor data.
* Integrated BLE HID functionality using the ESP32 for mouse communication.
* Understood how to use flex and touch sensors for alternative input methods.
* Gained experience in low-power design, schematic creation, and debugging hardware issues like GPIO conflicts and connection errors.
* Practiced real-world embedded systems concepts including power management, analog sensor reading, and user interface design.

## 7.2 FUTURE PROSPECTS

Enhancing your hand gesture-controlled mouse can take several directions, depending on your goals. Below is a detailed breakdown of possible future improvements categorized by accuracy, functionality, hardware, connectivity, and applications.

**Accuracy and Responsiveness Improvements**

To make the gesture-controlled mouse more precise and user-friendly, consider these advancements:

1. Sensor and Camera Enhancements

Higher Resolution Cameras: Switching to depth cameras (like Intel RealSense or Leap Motion) or LiDAR can provide better 3D tracking accuracy.

Infrared Sensors: Using IR-based tracking (like the Xbox Kinect) can help in low-light conditions.

Multiple Camera Setup: A stereo camera setup can provide better depth estimation for tracking hand movements.

1. Advanced Gesture Recognition Algorithms

Machine Learning Models: Implement Convolutional Neural Networks (CNNs) or Transformer-based models to improve gesture recognition.

Kalman Filters: Smooth out noise and fluctuations in gesture inputs.

Edge Detection & Optical Flow: Improve motion tracking using computer vision techniques.

1. Finger and Palm Tracking

Multi-finger tracking: Instead of tracking the entire hand, allow for finer control by recognizing individual fingers.

Palm Position & Rotation Detection: Determine hand orientation for additional control inputs (e.g., tilt-based scrolling).

**Additional Gesture Support and Customization**

Expanding gesture recognition makes the system more user-friendly and powerful.

1. Customizable Gesture Mapping

* Allow users to define their own gestures and map them to specific actions (e.g., opening apps, adjusting volume).
* Implement an AI-powered learning system that adapts to individual user habits.

1. Multimodal Interaction (Combining Inputs)

* Gesture + Voice Commands: Users can combine gestures with voice control for more flexibility.
* Eye Tracking Integration: Use eye movement for cursor positioning and gestures for actions.

1. Complex Gesture Sequences

* Recognize gesture combinations (e.g., swipe left + pinch to close a window).
* Add two-handed gestures (e.g., using both hands to zoom in/out).

**Hardware Enhancements**

Improving the physical design and integration with hardware components.

1. Wireless & Wearable Solutions

* Bluetooth/Wi-Fi Connectivity: Make the system wireless to improve usability.
* Wearable Device: Use a ring, glove, or wristband for gesture tracking without a camera (e.g., an IMU-based system like Myo Armband).

1. Haptic Feedback Integration

* Add vibrations or haptic feedback for confirmation when gestures are recognized.
* Implement a tactile response system using small actuators in a glove-based interface.

1. Energy Efficiency

* Implement low-power sensors to reduce battery consumption in wireless setups.
* Use energy-harvesting techniques (e.g., kinetic energy from hand movements).

**Connectivity and Integration with Other Devices**

Expanding the compatibility of the system.

1. Cross-Platform Compatibility

* Support for Windows, macOS, Linux, and mobile devices.
* Implement a browser-based interface for web applications.

1. Smart Home & IoT Integration

* Control smart home devices (e.g., lights, thermostat) with gestures.
* Integrate with virtual assistants (Alexa, Google Assistant) for multimodal interactions.

1. Augmented Reality (AR) & Virtual Reality (VR) Applications

* Use gestures to interact with AR/VR environments.
* Enable virtual object manipulation using 3D gesture tracking.

**AI-Powered Enhancements**

Using AI to improve prediction and interaction.

1. Predictive Actions

* AI can anticipate user intent based on previous actions.
* Example: If the user frequently opens a browser after starting the system, the AI can suggest or automate this task.

1. Adaptive Learning System

* The system should adapt to different users by learning their unique gestures.
* It can offer real-time gesture correction and suggest optimizations for better performance.

1. Real-Time Feedback & Virtual Assistant

* Implement an on-screen assistant that provides guidance when a gesture is misinterpreted.
* Example: If a user’s hand is not positioned correctly, the assistant can suggest adjusting the angle.

**Potential Use Cases and Applications**

Different industries and user needs can benefit from an enhanced gesture-controlled mouse.

1. Accessibility for Disabled Users

* Users with mobility impairments can benefit from customized gestures for easier control.
* Support head-tracking + hand gestures for a more accessible interface.

1. Gaming Enhancements

* Use gestures for in-game controls, especially in VR and AR gaming.
* Integrate with haptic gloves for a more immersive experience.

1. Professional & Industrial Applications

* Graphic designers & artists: Use pinch and zoom for drawing applications.
* Medical field: Surgeons can interact with medical images without touching a surface.
* Manufacturing: Workers can control machinery with simple hand movements.

By implementing better sensors, AI-based gesture recognition, multimodal input, and AR/VR compatibility, your hand gesture-controlled mouse can become more accurate, efficient, and widely applicable.

# CHAPTER 8

# REFERENCES

1. "A Quick History of Drag and Drop – A GoPhore Article". 365Trucking.com. Archived from the original on 2019-07-02. Retrieved 2019-07-02.
2. ^ "MX vs. ATV: Reflex PC UK Manual" (PDF). p. 3. Archived (PDF) from the original on 13 February 2022. Retrieved 13 February 2022.
3. ^ "Windows 7 Hardware: Touch Finally Arrives". 2009-09-28. Archived from the original on 2012-11-07. Retrieved 2012-11-19.
4. ^ Hancock, Mark; ten Cate, Thomas; Carpendale, Sheelagh (2009). "Sticky tools". Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces - ITS '09. New York, New York, USA: ACM Press. p. 133. doi:10.1145/1731903.1731930. ISBN 978-1-60558-733-2.
5. ^ "Opera Tutorials - Gestures". Archived from the original on 7 September 2012. Retrieved 3 August 2012.

* Buxton, W. A. (1995). "Chunking and phrasing and the design of human-computer dialogues" in Human-Computer interaction: Toward the Year 2000, R. M. Baecker, J. Grudin, W. A. Buxton, and S. Greenberg, Eds. Morgan Kaufmann Publishers, San Francisco, CA, 494-499.
* The Unknown History of Pen Computing Archived 2017-04-18 at the Wayback Machine contains a history of pen computing, including touch and gesture technology, from approximately 1917 to 1992.
* Annotated bibliography of references to handwriting recognition, gesture user interfaces, and pen computing[permanent dead link]
* L. K. Welbourn and R. J. Whitrow. 1988. A gesture based text editor. In Proceedings of the Fourth Conference of the British Computer Society on People and computers IV, D. M. Jones and R. Winder (Eds.). Cambridge University Press, New York, NY, USA, 363-371. ISBN 0-521-36553-8
* Brad A. Myers. "A Brief History of Human Computer Interaction Technology Archived 2019-06-18 at the Wayback Machine". ACM interactions. Vol. 5, no. 2, March, 1998. pp. 44–54.
* Notes on the History of Pen-based Computing (YouTube) Archived 2021-11-10 at the Wayback Machine.

**EXTERNAL LINKS**

1. Annotated Bibliography of Reference to Gestures, Touchscreens, and Pen Computing Archived 2013-12-15 at the Wayback Machine
2. Notes on the History of Pen-based Computing on YouTube
3. Notes on the (relatively unknown) History of Pen-based Computing Archived 2016-08-23 at the Wayback MachinE.

# CHAPTER 9

# APPENDICES

#include <Wire.h>

#include <MPU6050.h>

#include <BleMouse.h>

int filterIndex = 0;

#define SCROLL\_BUTTON\_PIN 33 // Button GPIO

#define FILTER\_SIZE 5 // Size of Moving Average Filter

#define BUTTON\_PIN 4 // GPIO 4 for right-click button

#define FLEX\_PIN 32 // GPIO 32 for flex sensor (analog input)

int xHistory[FILTER\_SIZE] = {0};

int yHistory[FILTER\_SIZE] = {0};

MPU6050 mpu;

BleMouse bleMouse("ESP32 Air Mouse", "ESP32", 100); // BLE HID Mouse

// Define threshold for flex sensor to trigger left-click

#define FLEX\_THRESHOLD 850 // Adjust this value based on your flex sensor's output

#define MIN\_FLEX\_VALUE 10 // Minimum flex value to avoid zero voltage (adjust as needed)

int movingAverage(int newVal, int history[], int size) {

history[filterIndex] = newVal;

filterIndex = (filterIndex + 1) % size;

int sum = 0;

for (int i = 0; i < size; i++) sum += history[i];

return sum / size;

}

void setup() {

Serial.begin(115200);

Wire.begin();

// Initialize MPU6050

mpu.initialize();

if (!mpu.testConnection()) {

Serial.println("MPU6050 connection failed!");

while (1);

}

Serial.println("MPU6050 Connected!");

// Start BLE Mouse

Serial.println("Starting BLE...");

bleMouse.begin();

Serial.println("BLE Started");

// Set up button as input with pull-up resistor

pinMode(BUTTON\_PIN, INPUT\_PULLUP);

// Set up flex sensor as input (analog pin)

pinMode(FLEX\_PIN, INPUT);

}

void loop() {

if (bleMouse.isConnected()) {

int16\_t ax, ay, az, gx, gy, gz;

mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);

//Scrolling Function

if (digitalRead(SCROLL\_BUTTON\_PIN) == LOW) { // Button pressed

int scrollDirection = 0;

if (gy > 1000) { // Tilt forward (scroll down)

scrollDirection = -1;

} else if (gy < -1000) { // Tilt backward (scroll up)

scrollDirection = 1;

}

if (scrollDirection != 0) {

bleMouse.move(0, 0, scrollDirection);

Serial.printf("Scrolling: %s\n", scrollDirection == 1 ? "Up" : "Down");

}

}

delay(50); // Adjust delay for responsiveness

// Convert gyro values to mouse movement

int xMove = gy / 1000; // Scale gyroscope values

int yMove = gx / 1000; // Invert Y-axis for natural control

// Apply moving average filter

xMove = movingAverage(xMove, xHistory, FILTER\_SIZE);

yMove = movingAverage(yMove, yHistory, FILTER\_SIZE);

// Apply dead zone to prevent jittering

if (abs(xMove) < 2) xMove = 0;

if (abs(yMove) < 2) yMove = 0;

// Move the mouse

bleMouse.move(xMove, -yMove);

Serial.printf("Mouse Move: X=%d, Y=%d\n", xMove, yMove);

// Check button state for right-click

if (digitalRead(BUTTON\_PIN) == LOW) { // Button is pressed

Serial.println("Right Click!");

bleMouse.click(MOUSE\_RIGHT);

delay(500); // Debounce delay to avoid multiple clicks

}

// Read the flex sensor value

int flexValue = analogRead(FLEX\_PIN);

// Check if the flex sensor value exceeds the threshold to trigger left-click

// Also ensure that flex value is above the MIN\_FLEX\_VALUE to avoid zero voltage

if (flexValue < FLEX\_THRESHOLD && flexValue > MIN\_FLEX\_VALUE) {

Serial.println("Left Click!");

bleMouse.click(MOUSE\_LEFT);

delay(500); // Debounce delay to avoid multiple clicks

}

delay(10); // Adjust delay for smoothness

}

}