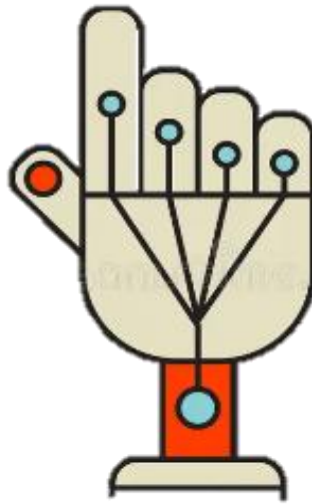




# Gesture control gaming glove

Embedded system project



بإشراف دكتور موسى اليمين

0222211	اسيا سامر	0223457	جنى صالح محمود غصية
0223839	ليلى اللحام	0222209	لين ماهر جرس المصاروة
0222126	ايثار فيصل فلاح السلامة	0224982	دانا معتصم أبو الرز

## Contents

Table of figures.....	4
1.0 Introduction:.....	8
2.0 Flowchart of the system operation:.....	8
3.0 Flowchart Description.....	10
3.1 Power On and System Initialization.....	10
3.2 Sensor Data Acquisition: .....	10
3.3 Data Transmission using UDP: .....	10
3.4 Data Processing in Python Script:.....	10
3.5 Gesture-to-Action Mapping: .....	11
3.6 Visual and Audio Feedback: .....	11
4.0 Description of Actuators and Transducers Used in the System.....	12
4.1 Actuators: .....	12
4.1.1. Laptop Screen (Visual Actuator):.....	12
4.1.2. Laptop Speakers (Audio Actuator):.....	12
4.1.3. Laptop (Processing and Output Device):.....	12
4.2 Transducers Used:.....	13
4.2.1. Flex Sensor Threshold Logic (Shooting Detection): .....	13
4.2.2. MPU6050 (for motion/orientation): .....	13
4.3 Supportive Components.....	14
4.3.1. ESP32 Microcontroller.....	14
4.3.2. Breadboard and Jumper Wires .....	14
4.3.3. 7.4V Li-ion Battery and Step-Down Regulator .....	14
4.3.4. Resistors (47k $\Omega$ ) .....	15
4.3.5. Sensor Mounting.....	15
5.0 System Architecture Overview: .....	15
6.0 Connections .....	15
7.0 System Expansion Potential: .....	17

8.0 Conclusion: .....	17
8.1. A Cheaper Option for Motion Control.....	17
8.2. New Opportunities in the Gaming Market.....	18
8.3. Can Be Used in Other Areas Too .....	18
8.4. Encouraging Innovation and Tech Skills.....	18
8.5. More Fun = More Spending.....	19
9.0 Final Thoughts .....	19

## Table of figures

Figure 1 ESP32 .....	5
Figure 2 MPU6050 .....	5
Figure 3 Flex sensor .....	5
Figure 4 resistor .....	5
Figure 5 Voltage reducor .....	5
Figure 6 Breadboard .....	5
Figure 7 Battery .....	6
Figure 8 Bread board wires .....	6
Figure 9 on/off switch .....	6
Figure 10 Glove .....	6
Figure 11 3D printed MPU6050 case .....	6
Figure 12 laptop with Python interface .....	6
Figure 13 Zomboid survival unity game .....	7



Figure 1 ESP32



Figure 2 MPU6050

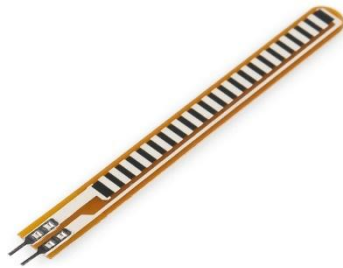


Figure 3 Flex sensor



Figure 4 resistor



Figure 5 Voltage reducer

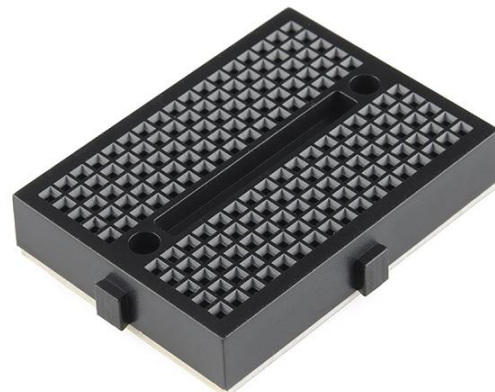


Figure 6 Breadboard



Figure 7 Battery

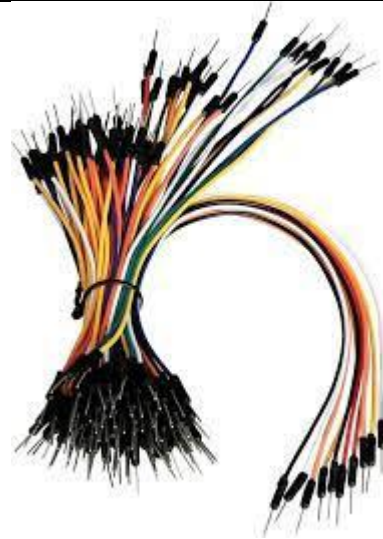


Figure 8 Bread board wires



Figure 9 on/off switch



Figure 10 Glove



Figure 11 3D printed MPU6050 case



Figure 12 laptop with Python interface



Figure 13 Zomboid survival unity game

## 1.0 Introduction:

In the current dynamic technological landscape, the advancement of contemporary user interfaces is becoming a crucial focus of research and design. Some prominent examples involve the development of interactive systems that improve and human–computer interaction (HCI) through gestures. In the realm of human–machine interaction, gesture recognition is seen as the most intuitive and natural approach. In this project we explore the development of a low-cost smart glove prototype designed to capture dynamic hand gestures for game control .This prototype has potential applications in various fields such as sign language translation gloves, controlling robotic hands, security technologies, and—our chosen focus—gaming gesture gloves.

Everybody likes to play games and there are many ways of playing them. We all know PlayStation, Xbox, PC and Nintendo and the regular game controllers. The shape may differ slightly from brand to brand, but in general they all look the same. We think it is time to design something new and expand the possibilities of your gaming experience!

An important factor that played a part in our project selection is including people with disability in the gaming experience. Think of people with only one working hand. Most things are hard for these people. One of these things is using the regular game controller. It is made for two hands! Can you imagine how awkward gaming must be with only one hand? We wanted to come up with a solution, that doesn't leave anyone behind.

A gaming glove that captures simple hand gestures and uses them to explore and control games for a more immersive and enjoyable gaming experience.

## 2.0 Flowchart of the system operation:

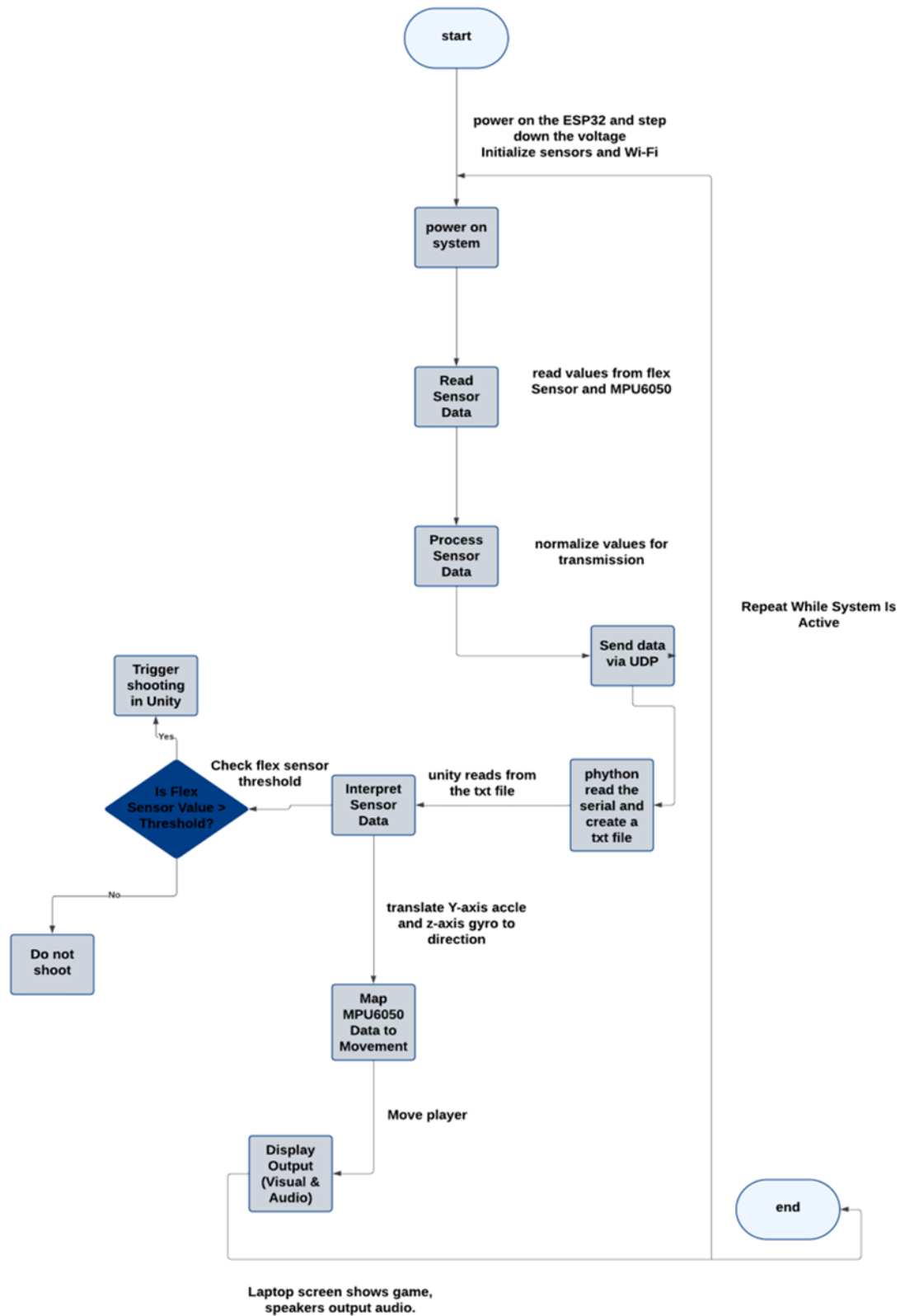


Figure 14 Flowchart of the system

## 3.0 Flowchart Description

### 3.1 Power On and System Initialization

- The system runs on a 7.4V Li-ion battery, which is regulated down to 5V by a voltage regulator to supply the ESP32. Once powered, the ESP32 initializes the necessary GPIOs and I2C communication with mounted sensors (flex sensor and MPU6050). It also establishes a UDP connection to the laptop.

### 3.2 Sensor Data Acquisition:

- The flex sensor, which is strapped to the index finger using a silicone glove, gives analog readings to the ESP32.
- The MPU6050, contained in a 3D-printed case in the glove's rear, gives the gyroscope readings (movement Y-axis, rotation Z-axis) constantly through I2C.
- The ESP32 reads these values in real time and prepares them to send.

### 3.3 Data Transmission using UDP:

- ESP32 transmits the sensor readings collected wirelessly to the laptop using UDP protocol. This ensures low-latency transmission for fast gameplay.

### 3.4 Data Processing in Python Script:

- The laptop hosts a Python script that reads serial output and writes it on a txt file that the Unity game code checks:
- If the value of the flex sensor goes over a shooting threshold.

- If MPU6050 values indicate forward/backward movement or left/right turning.

### 3.5 Gesture-to-Action Mapping:

- The Unity code interprets the sensor readings to respective game actions:
- **Finger fold** (flex sensor) → shoots the weapon.
- **Move forward/backward (MPU6050 Y-axis)** → moves the player forward/backward.
- **Turn left/right (MPU6050 Z-axis)** → turns the direction of the player.
- These commands are communicated to the Unity game environment.

### 3.6 Visual and Audio Feedback:

- The laptop screen (visual actuator) displays the game, and the laptop speakers (audio actuator) provide sound effects such as footsteps, guns, or zombie screams. These feedbacks are a result of the player's actions and maintain the player engaged.

### Continuous Loop:

The system runs in a continuous loop:

- Sensors continue to receive new input.
- Data is sent to the laptop.
- Python updates a txt file read by the Unity game code.
- The loop ensures the player gestures are consistently updated on the game without delays.

## 4.0 Description of Actuators and Transducers Used in the System

### 4.1 Actuators:

Our system doesn't use typical actuators; instead, it does use software-based actuators in order to deliver rapid input within the game environment. The sensors collect all user inputs and then convert them into responses in the Unity gaming engine, which allows real-time interaction without needing physical motion output devices.

#### 4.1.1. Laptop Screen (Visual Actuator):

The laptop screen functions as a visual actuator by displaying real-time visual feedback based on the user's actions, such as movement and shooting, making it an essential component for delivering visual output in our system.

#### 4.1.2. Laptop Speakers (Audio Actuator):

The game's audio (e.g., zombie sounds, gunfire, movement) provides auditory responses, helping the player stay immersed and react accordingly. These sounds are triggered by sensor input read through Python.

#### 4.1.3. Laptop (Processing and Output Device):

The laptop serves as the central processing and output unit of the system. It performs three main functions:

Receives input from the ESP32 via Wi-Fi using UDP protocol.

Runs a Python script that reads values from the flex sensor and the MPU6050 motion sensor and writes them on a txt file.

Controls the Unity-based zombie game by mapping the player's physical gestures to in-game actions such as movement and attacks.

## 4.2 Transducers Used:

To track player gestures and turn them into game actions, we used two main sensors that were both simple to work with and effective:

### 4.2.1. Flex Sensor Threshold Logic (Shooting Detection):

- To detect the shooting action, we implemented a threshold-based condition in the ESP32's code:
  - The analog value from the flex sensor ranges from 0 to 4095.
  - When the finger is straight, the value is high (close to 4000).
  - When the finger is bent, the value drops significantly.
  - We set a threshold around 3000 (after testing, we tried values like 200 or 400 but found 3000 worked best).
- This simple logic made it very responsive and allowed us to detect when the player bends their finger to shoot.
- The signal "shoot" was sent to the laptop via UDP, where the Python script writes it on a txt and then Unity reads the signal and fires a shot in-game.
- We chose this sensor because it's lightweight, simple to attach, and makes shooting feel like a natural finger gesture.

### 4.2.2. MPU6050 (for motion/orientation):

This small sensor combines an accelerometer and a gyroscope, allowing us to detect movement and direction. We placed it on the back of the player's hand, protected and stabilized inside a custom 3D-printed case.

- Reading the values:
  - The MPU6050 connects to the ESP32 using I2C protocol:
  - SDA to GPIO21
  - SCL to GPIO22
  - We added 47k $\Omega$  pull-up resistors to both lines for stable communication.
- The ESP32 uses an I2C library to get real-time data:

- Z-axis gyroscope: used for detecting left/right turning
- Y-axis accelerometer: used for detecting forward/backward motion
- This data is also sent via UDP to the laptop, and Python writes the readings on a txt file that the Unity code interprets and maps it into movement commands in the game.
- We chose this sensor because of its compact size, good accuracy, and ability to smoothly track hand gestures.

## 4.3 Supportive Components

To make the wearable system work smoothly, we used a few key components that helped us build and test everything easily:

### 4.3.1. ESP32 Microcontroller

This board is the brain of the system. It reads data from the sensors and sends it to the laptop via Wi-Fi using UDP protocol. We chose the ESP32 because it's powerful, has great connectivity options, and works well in small wearable setups.

### 4.3.2. Breadboard and Jumper Wires

We used a breadboard to avoid soldering during testing. Jumper wires made it easy to connect and change components quickly. This setup saved time while we were adjusting the circuit.

### 4.3.3. 7.4V Li-ion Battery and Step-Down Regulator

To keep the system portable, we used a 7.4V Li-ion rechargeable battery. Since this voltage is too high for the ESP32, we connected it to a step-down voltage regulator to reduce the voltage to 5V.

Instead of a battery holder, we used a custom connector piece to link the battery to the step-down regulator.

This setup removed the need to keep the system plugged in and allowed full wireless operation via Wi-Fi and UDP.

#### 4.3.4. Resistors (47kΩ)

For the flex sensor's voltage divider, we used a 47kΩ resistor, which gave us more stable and readable analog values during finger bending.

#### 4.3.5. Sensor Mounting

The flex sensor was fixed to a silicone glove, making it comfortable and secure for natural finger gestures.

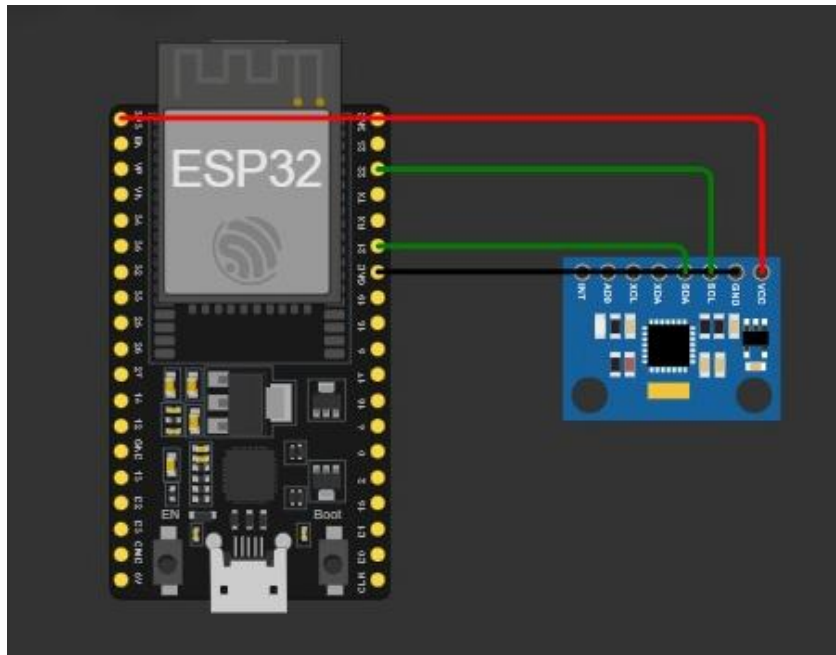
The MPU6050 was embedded in a 3D-printed case and attached to the glove's back for reliable motion detection.

## 5.0 System Architecture Overview:

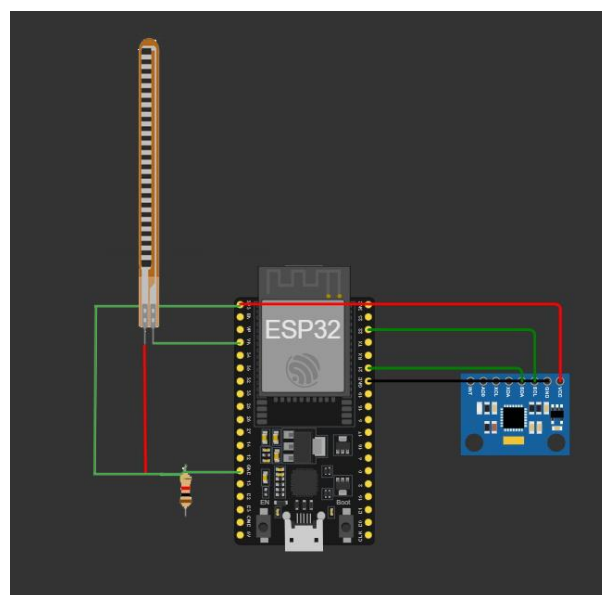
- Here's a simple breakdown of how everything works together:
  1. The sensors (flex and MPU6050) are connected to the ESP32.
  2. The ESP32 reads their data and sends it to the laptop wirelessly using UDP.
  3. On the laptop, a Python script reads the data and writes it to a txt file that is read by Unity.
  4. Unity interprets the readings into actions that are passed to the game, so the player's gestures show up immediately during gameplay.
- This setup makes it easy to improve or expand later on. For example, we can add more sensors or change how gestures are mapped—without needing to rebuild everything from scratch.

## 6.0 Connections

Component	Pin on Component	Connects to ESP32 Pin
<b>MPU-6050</b>	VCC	3.3V
	GND	GND
	SCL	GPIO 22
	SDA	GPIO 21



Component	Pin on component	Connects to ESP32 Pin
<b>Flex sensor</b>	One side	Directly to GPIO 34
	Other side	Soldered to a resistor(also connected to 3.3V);resistor's other end to GND



## 7.0 System Expansion Potential:

The modular design enables expansion. Additional gestures or sensors can be added by making adjustments in the Python logic and Unity mappings without altering the hardware setup.

**Battery Efficiency:** Incorporating a more powerful and energy-efficient battery system to extend operational time without increasing weight significantly.

**Sensor Enhancement:** different sensors can be added so the glove can include a wider range of games and replace WSAD controls

## 8.0 Conclusion:

In this project, we designed and built a gaming gesture glove that lets players control a game using hand movements. We tested it on a zombie-killer game to show how it works in a real gaming situation. While the main goal was to make gameplay more fun and interactive, our project also has the potential to make an economic impact.

### 8.1. A Cheaper Option for Motion Control

Many advanced gaming systems, like VR and motion sensors, can be very expensive and not affordable for everyone. Our glove uses low-cost components, which makes it a more affordable way to enjoy motion-controlled gaming. This could help more people try out this type of gaming experience without needing to spend a lot of money. That means it could be a good product for people who want something fun and new, but at a lower cost.

## 8.2. New Opportunities in the Gaming Market

The gaming industry is one of the biggest in the world, making billions of dollars every year. A product like our glove could open up new business opportunities. For example:

- Game developers could create games that are made just for glove use.
- Tech companies could improve and sell their own versions of the glove.
- Players might buy extra accessories to use with the glove.

So, this kind of project doesn't just make games more exciting—it also creates chances for new products and jobs.

## 8.3. Can Be Used in Other Areas Too

Even though the glove was made for gaming, it could also be useful in other fields. With a few changes, it could be used for:

- **Rehabilitation** to help people improve hand movement.
- **Simulation training** for things like safety drills or learning hand skills.
- **Education**, where students could interact with lessons using hand motions.

These other uses could make the glove valuable in areas like health, safety, and learning—not just in gaming. That means it could have a wider economic impact across different industries.

## 8.4. Encouraging Innovation and Tech Skills

Projects like this help students learn about sensors, hardware, and software development. It also helps local tech communities grow by encouraging young people to invent and build. If more students or startups work on similar ideas, it could lead to new businesses and tech jobs, especially in fields related to gaming and wearable technology.

## 8.5. More Fun = More Spending

When a game is more fun or different from others, people are more likely to play it longer or spend money on it. A glove like this adds a new way to play, which could lead to:

- Buying more games that support it
- In-game purchases or customizations
- Upgrades to the glove itself

This means the glove could help both game makers and sellers by increasing profits.

## 9.0 Final Thoughts

In the end, our controller glove shows how simple, affordable technology can lead to big ideas. It makes games more interactive and fun, while also having the potential to affect the economy in a positive way. Whether it's by helping new businesses grow, offering cheaper alternatives to expensive systems, or opening the door to new tech ideas, this project proves that even small innovations can make a big difference.