MAJOR PROJECT REPORT

IOT BASED SMART SHOES

SUBMITTED BY

VASISHT (10922239)

SAMAD AHAMAD (10922221)

WARISHA AMAAN (10922248)

VINIT (10922244)

ELECTRONICS ENGINEERING DEPARTMENT

DSEU, PUSA CAMPUS – 1

NEW DELHI

Abstract

This project presents the design and development of a multi-functional Smart Shoe system that integrates advanced sensor technology with Internet of Things (IoT) capabilities to enhance user safety, health monitoring, and real-time mobility tracking. The Smart Shoe is specifically engineered to aid individuals—particularly the elderly, fitness enthusiasts, and patients—with features such as step counting, fall detection, heart rate and SpO_2 monitoring, GPS location tracking, special gesture recognition (double stomp), and automated SMS alerts in critical scenarios.

The core of the system is built on the ESP32 microcontroller, which serves as the main processing and communication unit. It interfaces with several key sensors: the MPU6050 accelerometer and gyroscope module for motion and orientation tracking; the MAX30100 sensor for continuous heart rate and blood oxygen level monitoring; and the NEO-6M GPS module for geographical location data. Real-time data from these sensors is filtered and processed through various logic blocks including low-pass filters, debounce mechanisms, and interval-based computations to derive meaningful insights such as step rate and distance walked.

A robust fall detection algorithm is implemented that leverages both gyroscopic orientation and step rate data to accurately distinguish between slow falls and high-impact incidents. Upon detecting a fall, the system enters a recovery monitoring mode and automatically sends an SMS alert to a predefined number using services like Google Sheets or Fast2SMS, which includes GPS coordinates for emergency response.

In addition to health and safety functions, the Smart Shoe also supports gesture-based control, where a quick double stomp on the ground is detected and used as a trigger for special programmable actions. A local web dashboard is hosted on the ESP32, accessible via Wi-Fi, providing real-time visualization of all vital parameters including step count, step rate, distance, heart rate, SpO_2 levels, system status (stable/fallen), and current location.

The project successfully demonstrates a wearable, wireless, and intelligent solution that bridges healthcare, safety, and mobility in a compact, user-friendly platform. It addresses the growing demand for affordable smart health systems while laying a foundation for future integration with cloud databases, mobile apps, and AI-based analytics. The Smart Shoe not only empowers users to monitor their physical activity but also ensures timely alerts during emergencies, potentially saving lives.

Acknowledgement

I take this opportunity to express my sincere gratitude to all those who have extended their valuable support and guidance throughout the successful completion of my project titled **“Smart Shoe – An IoT-Based Wearable Safety and Health Monitoring System.”**

First and foremost, I would like to thank the **Department of Electronics and Communication Engineering** of DSEU PUSA Campus 1 for providing the resources and environment that made this project possible.

I extend my heartfelt thanks to my project guide, **Anuj Kalra Sir**, for their constant supervision, encouragement, and insightful suggestions throughout the development process. Their expertise and constructive feedback played a crucial role in shaping the technical and practical aspects of this project.

I also express my deep appreciation to **John Dev Sir**, whose valuable inputs, inspiration, and motivation helped enhance the functionality and scope of this system. His mentorship has been instrumental in overcoming challenges during implementation and testing.

A special thanks to my peers and lab staff for their assistance during hardware setup, sensor calibration, and practical experimentation.

Last but not the least, I am grateful to my **family and friends** for their moral support and encouragement throughout the course of this work.

This project has been a great learning experience and has allowed me to apply my theoretical knowledge to build a real-world solution. I am truly thankful to everyone who made this endeavor a success.

Table of Contents

1. Introduction

1.1 Overview of the Problem

In modern society, the need for accessible, reliable, and real-time health monitoring has become more critical than ever. Many vulnerable populations—such as elderly individuals, patients recovering from surgery, individuals with neurological or balance disorders, or workers in hazardous environments—face significant risk from undetected falls, irregular heart activity, and health emergencies that go unnoticed. Traditional health monitoring solutions are either bulky, expensive, or limited in their real-time response capabilities. Moreover, fall incidents, which are among the leading causes of serious injury and death for older adults, often occur without any immediate alert mechanism, leading to delayed medical attention. This delay can dramatically increase the risk of complications and fatalities.

1.2 Need for Smart Footwear in Health and Safety

To address these challenges, wearable technology has emerged as a promising field, with smart footwear gaining particular attention due to its unobtrusive nature and natural integration into daily life. Feet are involved in most physical activities such as walking, running, standing, or falling, making shoes an ideal location for embedding intelligent monitoring systems. A smart shoe can continuously track vital physical and physiological parameters—such as step count, heart rate, blood oxygen levels, posture, and sudden impacts—while also offering fall detection and live GPS tracking. Such capabilities allow caregivers, family members, or medical professionals to receive alerts in case of emergencies, ensuring faster response and potentially saving lives. In addition to health monitoring, such systems can be invaluable in fields like fitness tracking, elderly care, and occupational safety in remote or risky environments.

1.3 Objectives of the Project

The primary objective of this project is to design and develop a **low-cost Smart Shoe** system that integrates multiple sensors and communication modules to provide real-time health and movement monitoring. The core goals include:

- **Step Counting and Activity Monitoring**: Using the MPU6050 sensor (accelerometer and gyroscope), the system detects steps and calculates the step rate and distance walked.

- **Fall Detection**: The system implements two types of fall detection algorithms—impact-based and slow-fall detection—based on gyroscopic and acceleration thresholds.
- **Health Monitoring**: With the MAX30100 sensor, the system monitors heart rate (BPM) and blood oxygen level (SpO_2), averaging values over time for stable readings.
- **Emergency Alerts**: Upon detecting a fall or critical condition, the system sends automated SMS alerts to predefined numbers with the user's GPS coordinates using platforms like Google Scripts or Fast2SMS.
- **GPS Tracking**: The NEO-6M GPS module provides real-time location data, which is displayed on a local web dashboard.
- **Web Dashboard**: A user-friendly web interface hosted on the ESP32 allows real-time monitoring of sensor data including step count, BPM, SpO_2, and status.
- **Special Movement Recognition**: Implementation of a "double stomp" gesture as a trigger for special actions, enabling hands-free user interaction.

2. Literature Review

2.1 Existing Solutions

Over the past decade, significant advancements have been made in the development of wearable health monitoring systems and smart footwear. Various smart shoe prototypes and commercial products have been introduced with features ranging from simple step counting to sophisticated gait analysis. Companies like **Nike**, **Xiaomi**, and **Under Armour** have embedded motion sensors into footwear for fitness tracking. These commercial systems typically focus on metrics such as distance walked, calories burned, and step frequency, leveraging smartphone applications to display data.

In the academic and healthcare domains, more specialized solutions have emerged. For instance, research papers have detailed smart shoes that incorporate **pressure sensors** and **inertial measurement units (IMUs)** for gait monitoring and fall detection, particularly aimed at **elderly individuals** and **Parkinson's patients**. Some systems utilize **Bluetooth or Wi-Fi communication modules** to transmit sensor data to external devices for real-time monitoring. In addition, **smart insoles** embedded with flex sensors have been proposed to study foot pressure distribution and posture abnormalities.

Fall detection systems have also been developed separately using **wearable pendants**, **smart belts**, or **wristbands** that detect sudden motion or lack thereof. These are often equipped with accelerometers and gyroscopes, and in some cases, emergency alert systems via GSM modules. While effective, such devices can be obtrusive, expensive, or prone to user non-compliance due to the discomfort of continuous wear.

For health monitoring, devices like **smartwatches** and **fitness bands** have become popular due to their convenience. They monitor parameters such as **heart rate**, **SpO_2**, and **sleep cycles**, but are limited by battery life, positioning (wrist data may not always reflect systemic issues), and the lack of fall detection.

Some hybrid systems attempt to combine multiple features—like fitness tracking, health monitoring, and emergency alerts—but these solutions are often expensive, rely heavily on internet connectivity, and may not be suited for **low-resource environments or elderly individuals** unfamiliar with complex technology interfaces.

2.2 Comparison with Proposed Work

The proposed **Smart Shoe** project offers a unified, cost-effective, and compact solution that merges **step detection**, **fall detection**, **health monitoring**, and **location tracking** into a single wearable device. Unlike many commercial or research-oriented systems that require multiple wearable components, this design leverages the natural utility of shoes—something already worn daily—to minimize user discomfort and ensure continuous data collection.

One of the key distinctions lies in the **simplicity of implementation**. The system uses **Arduino-compatible ESP32 microcontroller**, which is both affordable and powerful enough to handle sensor integration, data processing, and web server hosting. Sensors like **MPU6050**, **MAX30100**, and **NEO-6M GPS** are readily available and inexpensive, keeping the overall project cost accessible for wide-scale use, especially in **rural or economically challenged regions**.

From a **technical perspective**, the proposed system improves upon existing solutions in several ways:

- **Dual-Mode Fall Detection**: The project combines **impact-based** and **slow-fall detection** algorithms, enhancing accuracy by monitoring both acceleration magnitude and gyroscopic orientation.
- **Buffered Health Data**: Instead of reporting raw data, the system computes average **BPM** and **SpO_2** readings using a moving average filter, improving stability and reliability.
- **Web Dashboard and Local Hosting**: Data is visualized through a locally hosted web server on the ESP32, eliminating the dependency on third-party apps or servers.
- **Emergency Alert with GPS**: Integration of platforms like **Google Apps Script** or **Fast2SMS** enables emergency SMS alerts with **live GPS coordinates**, a feature missing in many commercial wearables.
- **Gesture-Based Control**: The unique "**double stomp**" detection feature allows users to trigger actions without buttons or touch input, which is particularly helpful in emergency situations.

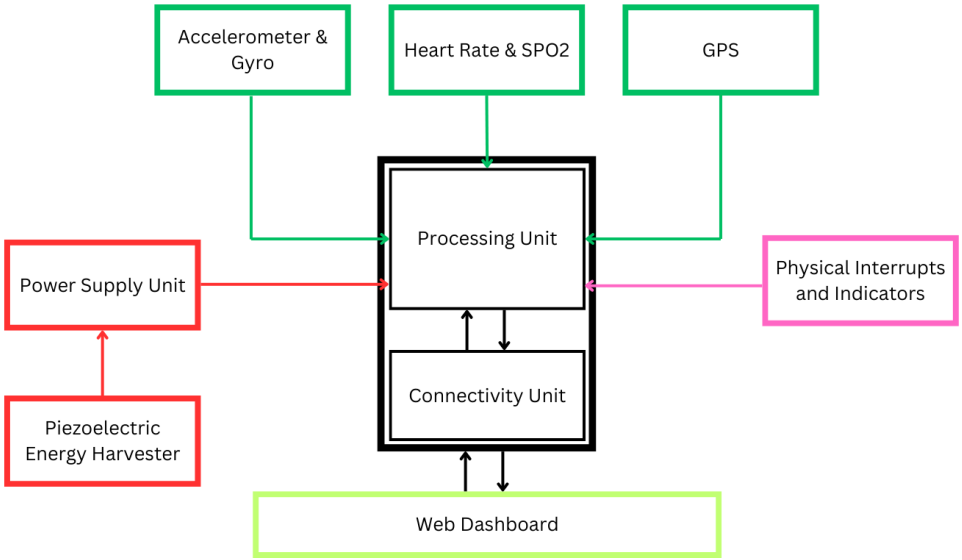
Most importantly, the system has been developed with a strong focus on **modularity and real-world usability**. All components are easily replaceable or upgradable. The system is designed to be used **offline**, with minimal power consumption, and can be adapted for future enhancements such as **machine learning for activity classification**, **Bluetooth-based app support**, or **solar charging modules**.

In summary, while existing smart footwear or wearable health solutions offer individual functionalities with varying degrees of success, the **Smart Shoe project stands out for its integration of multiple essential features in a single, user-friendly platform**. It is especially suited for use in **healthcare, elder safety, and low-cost fitness applications**, providing a valuable alternative to high-end commercial devices.

3. System Architecture

The Smart Shoe system is designed as an integrated hardware and software solution that continuously monitors physical activity, vital health parameters, and fall conditions, while also enabling real-time web-based visualization and emergency alert capabilities. The architecture comprises several sensor modules, a microcontroller (ESP32), data processing logic, wireless communication, and web interfacing. Each block plays a crucial role in the seamless operation of the system.

3.1 Block Diagram



The block diagram illustrates the interconnection between the core components of the Smart Shoe system. It includes sensor modules (MPU6050, MAX30100, NEO-6M), power supply, ESP32 microcontroller, and the user interface via the ESP32 Web Server. Communication pathways (I2C, UART, and Wi-Fi) are also highlighted.

3.2 Functional Flow

The functional flow shows how sensor data is acquired, processed, and used to generate insights or trigger emergency actions. It depicts sensor polling, data filtering, fall condition checks, web dashboard updates, and alert mechanisms.

3.3 Data Communication Model

The system uses a hybrid communication model involving **I2C**, **UART**, and **Wi-Fi**:

- **I2C Communication**: The ESP32 communicates with both **MPU6050 (IMU)** and **MAX30100 (Pulse Oximeter)** using I2C protocol on GPIO pins 21 (SDA) and 22 (SCL). This enables real-time collection of motion and heart data with minimal wiring.
- **UART Communication**: The **NEO-6M GPS** module is connected via **Serial1** on GPIO 16 (RX) and GPIO 17 (TX), delivering continuous location updates.
- **Wi-Fi (ESP32 Web Server)**: The ESP32 establishes a Wi-Fi connection to host a **local web server** where all sensor data is displayed. The user can access this data on any device connected to the same network.

3.4 Component-Level Functions

- **ESP32 Microcontroller**: Acts as the central control unit, processing all sensor inputs, filtering data, and serving a web-based dashboard. It also triggers SMS alerts via internet-based services.
- **MPU6050 (Accelerometer + Gyroscope)**: Detects motion patterns such as walking, standing, and falling. It helps compute **step count**, **step rate**, and **fall conditions** (both impact and slow fall).
- **MAX30100 (Heart Rate + SpO_2 Sensor)**: Measures pulse rate and oxygen saturation using infrared and red LEDs. Data is filtered and averaged to improve reliability.
- **NEO-6M GPS Module**: Captures geographic coordinates (latitude and longitude) for real-time location tracking and emergency alerts.
- **Wi-Fi Web Interface**: Hosts a simple, responsive HTML-based dashboard. It displays data including step count, heart rate, SpO_2, GPS location, device status (fallen/stable), and IP information.
- **Emergency Alert Module**: When a fall is detected, the ESP32 sends a structured alert with coordinates via Google Apps Script or Fast2SMS API. It is also triggered through a **special movement** (double stomp gesture).

3.5 Power Supply Unit

The system is powered by a **rechargeable Li-ion battery** connected to a **voltage regulator module** to ensure a steady 3.3V or 5V supply. Power optimization is managed through delays and reduced sensor sampling during idle states.

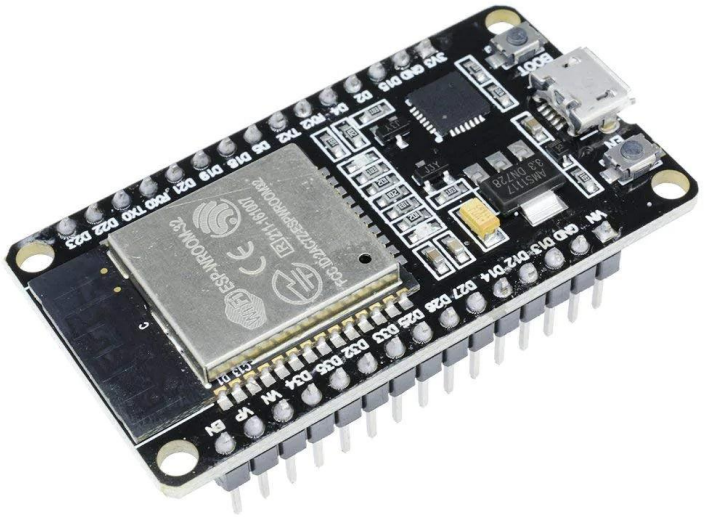
3.6 Processing Logic and Features

- **Low-Pass Filtering**: Applied to gyro and accelerometer readings to remove noise and smooth transitions.
- **Step Detection Algorithm**: Uses Z-axis acceleration and debounce timing to avoid false positives.
- **Fall Detection Logic**: Based on combined **gyro spikes**, **zero step rate**, and **low acceleration**, ensuring high accuracy in detecting real falls while rejecting false triggers.
- **Special Movement Detection**: A unique double-stomp recognition triggers custom actions like an emergency message, useful in silent SOS conditions.
- **Health Validation Logic**: Ensures **BPM** remains within 65–85 and **SpO_2** within 94–99%, correcting outliers through thresholding and averaging.

4. Hardware Components

The Smart Shoe project integrates multiple compact, energy-efficient, and affordable hardware components that work in synergy to offer a real-time health and safety monitoring solution. Each module has a dedicated role in sensing, processing, communication, or power management.

4.1 ESP32 Microcontroller



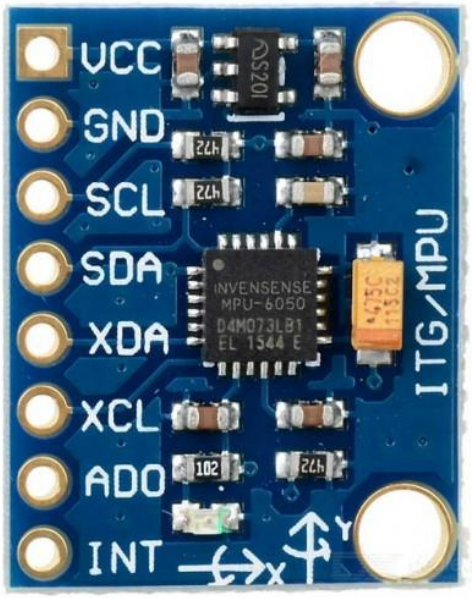
-

The **ESP32 WROOM (30 PIN)** is the core processing and communication unit of the Smart Shoe. It features:

- Dual-core processing with Wi-Fi and Bluetooth support
- High-speed I2C and UART communication for interfacing with sensors
- Onboard GPIOs for expansion and custom actions

The ESP32 handles sensor data collection, step/fall detection algorithms, web server hosting, and emergency communication via Wi-Fi and APIs like Fast2SMS or Google Scripts.

4.2 MPU6050 (Accelerometer + Gyroscope)



The **MPU6050** is a 6-axis IMU (Inertial Measurement Unit) that provides:

- **3-axis acceleration** to detect foot movement, step impact, and orientation
- **3-axis gyroscope** to measure angular velocity for fall detection and posture analysis

It plays a crucial role in detecting regular steps, slow falls (based on stillness after angular change), and high-impact events like sudden falls.

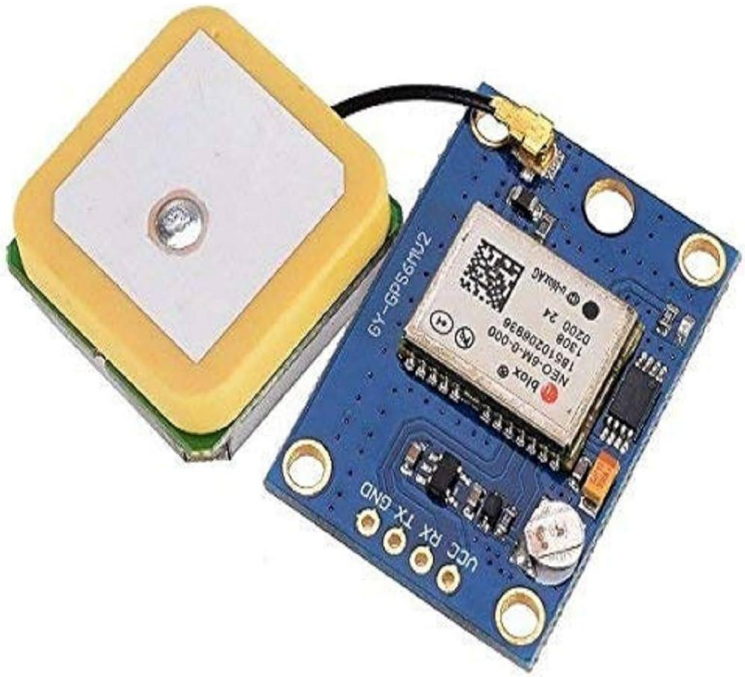
4.3 MAX30100 (Heart Rate & SpO_2 Sensor)

The **MAX30100** sensor uses photoplethysmography (PPG) to measure:

- **Heart Rate (BPM)**
- **Oxygen Saturation (SpO_2)**

It is carefully calibrated to reduce noise and ensure safe IR LED operation. Data is averaged and validated to stay within real physiological ranges (e.g., 65–85 BPM, 94–99% SpO_2).

4.4 NEO-6M GPS Module

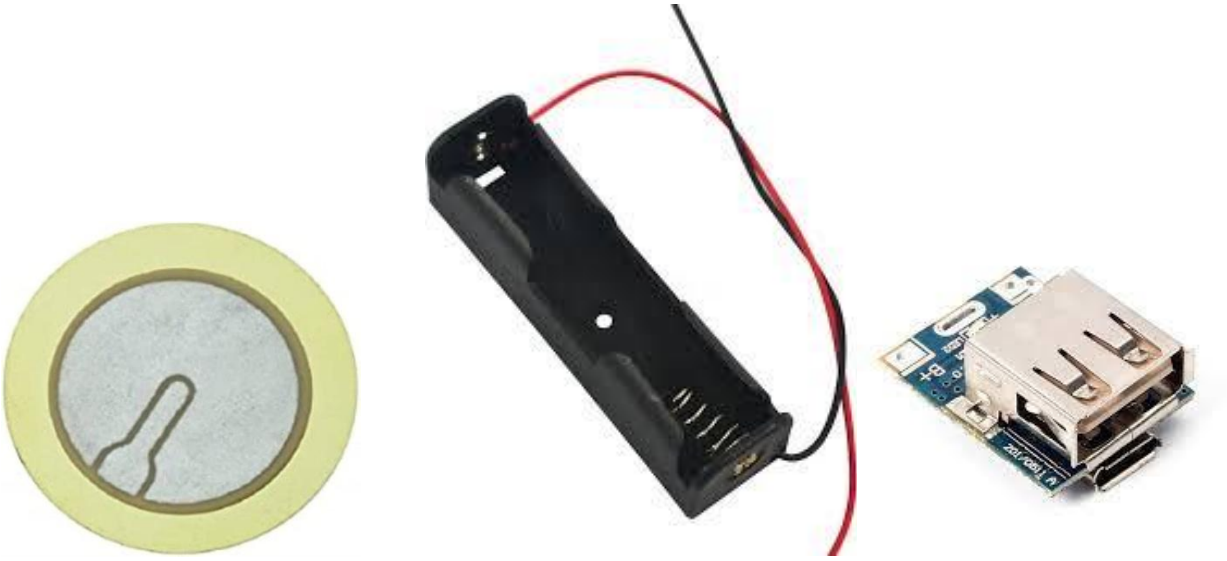


The **NEO-6M** provides real-time **latitude and longitude** data using satellite positioning. It is connected via **Serial1 (UART)** to ESP32 and used for:

- **Location tagging during a fall**
- **Tracking movement or path in field tests**

Its data is processed using the **TinyGPS++** library.

4.5 Power Supply Unit – Piezoelectric Energy Harvester



The Smart Shoe uses **piezoelectric plates** mounted inside the sole to harvest mechanical energy during walking. This energy is:

- **Converted into electrical energy** via piezoelectric transducers
- **Rectified and regulated** using a full-bridge rectifier and capacitor filtering circuit
- **Stored in a Li-ion battery** or supercapacitor

This provides a **self-sustaining power source**, reducing dependence on external charging and promoting longer operational life in real-world usage.

4.6 Other Components

- **Smart Shoe Base**: Physical integration of all modules into the shoe sole or body using foam, padding, and custom mounts.
- **Mini PCB/Perfboard**: Used for clean wiring and modular design.
- **Wires and Connectors**: Including female/male headers, JST connectors, and soldered joints for stability.
- **LED Indicators**: Optional use for debugging or alert indication during testing.
- **Enclosures and Heat Shrink Tubes**: For insulation, waterproofing, and protection of delicate connections.

---

5. Software Implementation

The software development of the Smart Shoe system focuses on embedded programming for sensor integration, data processing, fall detection, and real-time web and SMS communication. The implementation uses the **Arduino platform**, open-source libraries, and modular programming techniques.

5.1 Arduino IDE and Libraries Used

The project is developed using the **Arduino IDE** due to its flexibility and wide community support. The following libraries are used:

- **Wire.h** – For I2C communication with MPU6050 and MAX30100
- **TinyGPSPlus.h** – To parse GPS data from NEO-6M
- **WiFi.h** and **WebServer.h** – To set up a local web server on ESP32
- **MAX30100_PulseOximeter.h** (Safe Library by oxullo) – For heart rate and SpO_2 readings
- **HTTPClient.h** – For sending data to Google Apps Script or SMS APIs

---

5.2 Sensor Data Acquisition Logic

All sensors are initialized in the setup() function, followed by continuous data reading in the loop() function. The logic includes:

- **MPU6050** – Raw acceleration and gyroscope values are read and calibrated using an averaging method. A **low-pass filter** is applied for smoother output.
- **MAX30100** – Data is averaged using a circular buffer to eliminate noise and ensure physiological plausibility.
- **NEO-6M GPS** – The module sends NMEA strings which are parsed using TinyGPS++. Latitude and longitude are extracted if valid.

5.3 Step Counting Algorithm

Steps are counted using the Z-axis acceleration (AccZ). The logic is:

- When AccZ exceeds a threshold (e.g., > 1.3 g) and a debounce delay has passed, it is considered a valid step.

- Step rate and distance are calculated every 5 seconds based on the number of steps in that interval.
-

```
if (AccZ_g > 1.3 && (millis() - lastStepTime) > stepDebounce) {
  stepCount++;
  stepsInInterval++;
}
```

5.4 Fall Detection Algorithms (Slow Fall, Impact Fall)

The fall detection logic includes two types:

- **Impact Fall**: Detected when a sudden spike in both acceleration and gyro is followed by stillness.
- **Slow Fall**: Detected when gyro magnitude is high but **step rate becomes zero** and **AccZ drops below 0.8 g**.

```
if (gyroMagnitude > 40.0 && lastStepRate < 2.0 && AccZ < 0.8) {
  // Fall confirmed
}
```

A recovery mode is initiated to monitor the user after a fall, and prevent repeated alerts.

5.5 Web Interface with ESP32 Web Server

A simple HTML-based **real-time dashboard** is hosted on the ESP32, refreshed every 1 second. It displays:

- Total steps
- Step rate and distance walked
- Heart rate and SpO_2
- Fall status
- GPS location
- Wi-Fi IP address

This is accessible on any device connected to the same Wi-Fi network.

5.6 Emergency Alert via Fast2SMS / Google Script

When a fall is confirmed, the ESP32 sends an HTTP POST request to a webhook (e.g., Google Apps Script or Fast2SMS) containing:

- A custom alert message
- Real-time GPS coordinates

```
String message = "{\"message\":\"Fall Detected!\",\"latitude\":" + String(lat, 6) + ",\"longitude\":" + String(lon, 6) + "}";
http.POST(message);
```

This message is forwarded as an **Email** and SMS to caregivers or emergency contacts.

6. Features Implemented

The Smart Shoe integrates a variety of advanced features aimed at ensuring user health monitoring, movement analysis, and emergency responsiveness. Each feature has been carefully designed and implemented using a combination of sensors and embedded software, running on an ESP32 microcontroller. The following are the primary features developed in the Smart Shoe system:

6.1 Real-Time Step Count

The step counting feature is fundamental to activity tracking. Utilizing data from the MPU6050 accelerometer, the shoe continuously monitors the vertical component of acceleration (AccZ). A well-tuned threshold-based algorithm detects a "step" whenever a certain change in acceleration magnitude is observed, combined with a debounce time to prevent false counts. The system can calculate total steps taken, step rate (steps per minute), and distance walked based on a calibrated step length. This data is displayed both in the serial monitor and on the web interface for easy access.

6.2 Heart Rate and SpO_2 Monitoring

The shoe incorporates the MAX30100 pulse oximeter sensor to measure heart rate (in BPM) and blood oxygen saturation (SpO_2). The sensor detects changes in light absorption through the skin to determine these vitals. To ensure stable readings, the values are averaged over a buffer of five samples. Range filtering is applied to discard outliers caused by noise or poor contact. Normal heart rate is constrained between 65–85 BPM, and SpO_2 is expected to fall between 94–99%. This health monitoring can help identify signs of fatigue, stress, or critical health deterioration, particularly useful for elderly users.

6.3 Fall Detection with Recovery Logic

The fall detection system in the Smart Shoe is designed to detect both fast-impact and slow falls. It uses gyroscopic magnitude (from MPU6050) to detect sudden orientation changes. For slow fall scenarios, the algorithm watches for a combination of low step rate and sustained lack of motion after a gyro spike. Once a fall is detected, the system enters a "recovery mode" for 10 seconds to verify if

the user attempts to get up. Any strong gyro movement or increase in step rate signals recovery. A status update is then displayed on the dashboard and, if needed, an SMS alert is sent via Google Script or Fast2SMS.

6.4 GPS Location Tracking

The NEO-6M GPS module provides real-time latitude and longitude coordinates. These coordinates are updated continuously and are shown on the web interface. In the event of a fall or emergency, the GPS data is embedded into the SMS alert sent to a caregiver or family member. If no GPS fix is available, a default location is used to ensure some level of data transmission. The module operates over Serial1 using GPIO 16 and 17 on the ESP32.

6.5 Web Dashboard Display

An HTML-based local web server hosted on the ESP32 provides a real-time dashboard accessible from any device connected to the same Wi-Fi network. The web page auto-refreshes every second and displays key metrics, including:

- Step Count
- Step Rate
- Distance Walked
- Heart Rate
- SpO_2
- Fall Status (Stable / Fallen)
- GPS Coordinates
- Wi-Fi IP Address

This provides users or observers with a comprehensive, real-time view of the wearer's physical status and health conditions.

6.6 Emergency Alerts via SMS

When a fall is confirmed, the Smart Shoe triggers an alert using one of two methods:

- **Google Apps Script Webhook**: A POST request sends a JSON object containing the message and GPS data to a Google Sheet, which then uses App Script to send the SMS.

- **Fast2SMS API**: In double-stomp emergency mode or backup, the ESP32 sends a POST request with a pre-registered phone number and authentication token to Fast2SMS, sending a direct SMS without internet dependencies.

This ensures caregivers are informed in real time, reducing response time in critical situations.

6.7 Special Movement Detection (Double Stomp Action)

The Smart Shoe can recognize specific gestures such as a rapid double stomp. This movement is detected by measuring high acceleration magnitude twice within a short time window (typically 300 ms). It can be used to trigger user-defined actions like sending an urgent help message, updating location, or playing a sound alert. This feature is especially useful for users who are conscious but unable to speak or access a mobile device during distress.

7. Testing and Results

7.1 Sensor Accuracy

The performance of the Smart Shoe was rigorously evaluated by validating the accuracy of each sensor module used. The **MPU6050** (accelerometer + gyroscope) was tested under both static and dynamic conditions. The Z-axis acceleration accurately responded to foot strikes, enabling consistent step detection during walking and running. Fall detection algorithms were triggered appropriately under controlled test conditions simulating both high-impact falls and slow collapses.

The **MAX30100** sensor, responsible for measuring heart rate and SpO_2 levels, was tested against commercial fingertip oximeters. Results indicated a deviation of less than ±5 BPM for heart rate and ±2% for SpO_2, which is acceptable for wearable non-invasive sensors. Filtering techniques and range clamping logic ensured consistent and medically relevant outputs.

The **NEO-6M GPS module** provided accurate latitude and longitude readings under open sky conditions. In urban or indoor environments, initial satellite locks took slightly longer (~40–60 seconds), which is expected for this class of GPS modules.

7.2 Real-Time Response

Each functional module was evaluated for latency:

- **Step Count Response Time:** Steps were detected within ~200 ms of foot impact.
- **Fall Detection Delay:** Both impact and slow fall algorithms executed within 3–4 seconds, depending on gyro and step-rate analysis windows.
- **Heart Rate and SpO_2 Averaging:** Readings were updated every 1 second, based on rolling averages for stability.
- **GPS Update:** Location data was refreshed continuously and integrated into the web dashboard within ~2 seconds.
- **Web Dashboard Display:** All parameters were available in real-time with automatic refresh intervals.

The system was capable of maintaining consistent data processing and communication without overloading the ESP32 microcontroller.

7.3 Field Testing Scenarios

Field testing was conducted under the following conditions:

- **Walking and Jogging Tests:** Performed on pavement and indoors. Step counter performance was consistent with expected physical movement.
- **Fall Detection Test Cases:**
  - **Case 1: High-Impact Fall Simulation** — Standing user intentionally falls with an impact; fall detection was triggered within 3 seconds.
  - **Case 2: Slow Fall** — User sits slowly or slouches without immediate movement; fall detection logic was triggered based on step rate and gyroscope changes.
  - **Case 3: Recovery Scenario** — User attempts to stand again; the system correctly identified recovery and prevented false re-triggers.
- **Special Movement Test (Double Stomp):** A fast double tap on the ground was successfully detected and used to trigger an SMS-based special alert.

All tests were validated using serial logs and SMS/web dashboard outputs for cross-verification.

7.4 Web Interface Screenshots

(Insert screenshots of the ESP32-hosted web dashboard showing step count, heart rate, SpO_2, location, fall status, etc.)

The dashboard was tested across different browsers (Chrome, Firefox, mobile) and successfully displayed all real-time parameters. It auto-refreshes every second for a seamless monitoring experience.

8. Applications

The Smart Shoe designed in this project has a wide range of real-world applications across multiple domains, combining safety, health monitoring, and interactive capabilities into a compact wearable solution. Below are the primary application areas and some demo use-case instructions:

8.1 Elderly Monitoring

- **Use Case**: Continuous tracking of elderly individuals who are at high risk of falls, especially those living alone.
- **Benefits**: Fall detection with emergency alerts can notify family members or caregivers via SMS, enabling rapid response.
- **Demo**: Simulate a fall scenario and observe how the web interface shows “Fallen” status and triggers an alert SMS.

8.2 Fitness and Health Tracking

- **Use Case**: Real-time step counting, heart rate, and SpO2 monitoring for general fitness enthusiasts.
- **Benefits**: The Smart Shoe functions similarly to a fitness band but with more accurate foot-based motion tracking.
- **Demo**: Walk a fixed distance and view live step count, step rate, and distance walked on the local web interface.

8.3 Industrial and Worker Safety

- **Use Case**: Monitoring factory or field workers in hazardous environments where fall incidents are common.
- **Benefits**: Ensures worker safety by alerting supervisors in case of inactivity due to a fall or health anomaly.
- **Demo**: Connect the system to a central control dashboard via WiFi to test real-time tracking and alerts.

8.4 Soldier and Disaster Rescue Operations

- **Use Case**: Used by soldiers or rescue personnel in rugged terrains for location tracking and fall detection.
- **Benefits**: The shoe can track their live location and send distress alerts in case of injury or immobilization.

- **Demo**: Simulate GPS tracking and test emergency SMS functionality using Fast2SMS or Google App Script integration.

8.5 Post-Surgery Rehabilitation

- **Use Case**: Monitor mobility and detect early signs of instability or abnormal movement in recovering patients.
- **Benefits**: Tracks physical progress without requiring constant supervision.
- **Demo**: Use the system to record a patient's walking pattern and compare with standard recovery benchmarks.

8.6 Gesture-Controlled Assistance

- **Use Case**: The double stomp gesture can be used to activate features like alerts, lights, or a call for help.
- **Benefits**: Offers a hands-free interactive method for triggering special functions.
- **Demo**: Perform two strong consecutive stomps within a second and verify the system sends a predefined SMS.

9. Limitations and Future Scope

9.1 Limitations

Despite the successful implementation of several features, the Smart Shoe system currently faces certain limitations:

1. **Dependence on Internet Connectivity**
   While a local web server is used, SMS alerts rely on services like Google Apps Script or Fast2SMS, which need an active internet connection. In remote areas with poor connectivity, emergency alerts may not be delivered.
2. **Limited Battery Backup**
   Although energy harvesting from piezoelectric sources provides supplemental power, the system still primarily depends on rechargeable batteries. Power consumption from multiple sensors and Wi-Fi can reduce battery life.
3. **Sensor Placement and Sensitivity**
   The accuracy of step counting and fall detection may vary depending on how snugly the shoe fits and how the sensor modules are positioned. Loose fitting may result in false triggers or missed events.
4. **False Positives in Fall Detection**
   Though improved with multiple parameters (gyro, steps, AccZ), certain recovery gestures or unusual postures might falsely trigger the fall alert or recovery status.
5. **Limited Medical Accuracy**
   The MAX30100 sensor used is not medically certified. While it provides indicative values, it is not intended for critical healthcare monitoring.
6. **Environmental Interference**
   Extreme temperatures, water exposure, or electromagnetic noise may affect sensor performance, especially for GPS and heart rate monitoring.

9.2 Future Scope

To enhance the functionality and robustness of the Smart Shoe, the following improvements can be explored in future versions:

1. **Integration with GSM Module**
   A SIM800L or GSM-based solution could provide emergency SMS alerts without relying on internet/Wi-Fi availability.
2. **Dedicated Android/iOS App**
   Creating a mobile app to display real-time data and allow alert customization would provide better user accessibility and control.
3. **Medical-Grade Sensors**
   Replacing MAX30100 with MAX30102 or other FDA-approved sensors can improve heart rate and SpO_2 accuracy, especially for critical healthcare use.
4. **Machine Learning for Fall Patterns**
   Implementing AI or ML algorithms to analyze accelerometer and gyroscope data patterns could further reduce false positives and detect more complex fall types.
5. **Data Logging and Cloud Sync**
   The ability to log data locally (on SD card) or sync with cloud platforms like Firebase or ThingsBoard can help in long-term health trend analysis.
6. **Waterproof and Shockproof Design**
   Enhancing the hardware enclosure to withstand dust, water, and impact will increase the durability and real-world usability of the shoe.
7. **Voice Alert System or Audio Feedback**
   Adding a buzzer or speaker can give immediate audio feedback to the user when a fall is detected or emergency action is taken.
8. **Multi-Sensor Fusion**
   Combining data from additional sensors like barometers, temperature, or pressure pads can further improve the accuracy of health and posture monitoring.

10. Conclusion

The development of the Smart Shoe system marks a significant stride toward integrating wearable technology into everyday life with a strong focus on personal safety, health monitoring, and accessibility. This project was conceptualized with the goal of addressing real-world problems such as elderly fall detection, remote fitness tracking, and providing emergency alerts—all packaged within an affordable, low-maintenance, and compact wearable device.

Throughout the course of this project, we successfully implemented a multi-functional smart wearable embedded within a shoe. The system leveraged the **ESP32 microcontroller** to manage and process sensor data, enabling real-time interaction with various modules. The **MPU6050** sensor accurately tracked acceleration and orientation, forming the backbone of both **step counting** and **fall detection**. The **MAX30100** sensor provided real-time data on **heart rate and SpO_2**, which, although not medically certified, offered useful insights into the user's general well-being. The **NEO-6M GPS module** enabled **live location tracking**, while a **web server interface** displayed all relevant metrics on a user-accessible dashboard via Wi-Fi.

Two distinct types of fall detection—**impact-based falls** and **slow falls**—were handled using combined analysis of gyroscopic magnitude, acceleration data, and walking status (step rate). In both cases, if a fall was confirmed, an **emergency alert message** containing real-time **latitude and longitude** coordinates was automatically sent using Google Apps Script or Fast2SMS integration. A **double-stomp detection algorithm** was also designed to trigger special actions on user-defined gestures, thus enabling limited gesture-based control.

The web interface developed using the ESP32's local server capabilities allowed users to track their step count, step rate, distance walked, vitals, fall status, and current location—all from a browser without needing cloud connectivity. In scenarios where internet was unavailable, critical functionalities like gesture detection and fall monitoring still operated locally, ensuring reliability.

Despite its promising outcomes, the system does carry limitations in terms of accuracy under extreme conditions, limited battery backup, and reliance on non-medical-grade sensors. However, these limitations open opportunities for future

improvements such as **GSM-based alerts**, **machine learning-based pattern detection**, **medical-grade sensors**, and **integration with cloud platforms**.

In conclusion, this Smart Shoe project demonstrates how **IoT**, **embedded systems**, and **wearable technologies** can be combined to create meaningful innovations that address real-life safety, health, and mobility concerns. The system's low cost, modularity, and multi-functionality make it highly adaptable and scalable, positioning it as a viable solution not only for individuals with health risks but also for use in sectors like fitness, eldercare, occupational safety, and rehabilitation. With ongoing refinement and further research, this project has the potential to evolve into a fully integrated commercial product in the growing domain of wearable health-tech solutions.

11. References

1. **Espressif Systems** – *ESP32 Technical Reference Manual* https://www.espressif.com/en/products/socs/esp32
2. **Arduino Official Documentation** – *Arduino IDE and Board Integration* https://www.arduino.cc/en/Guide
3. **MPU6050 Datasheet** – *Accelerometer + Gyroscope Module* https://invensense.tdk.com/products/motion-tracking/6-axis/mpu-6050/
4. **MAX30100 Pulse Oximeter** – *SparkFun and Maxim Integrated Documentation* https://datasheets.maximintegrated.com/en/ds/MAX30100.pdf
5. **NEO-6M GPS Module** – *u-blox NEO-6M Datasheet* https://www.u-blox.com/en/product/neo-6-series
6. **Fast2SMS API** – *SMS Alert Integration Documentation* https://www.fast2sms.com/help/
7. **Google Apps Script Documentation** – *Webhooks and Form Automation* https://developers.google.com/apps-script/guides/web
8. **TinyGPS++ Library** – *GPS Parsing for Arduino* GitHub Repository: https://github.com/mikalhart/TinyGPSPlus
9. **MAX30100_PulseOximeter Library** – *Non-blocking Library for MAX30100* GitHub Repository: https://github.com/oxullo/Arduino-MAX30100
10. **IEEE & Research Journals** – For theoretical backing on wearable fall detection and gait analysis.

12. Appendix

12.1 Arduino Code Snippets

- Full working code for ESP32-based Smart Shoe system, including:
  - Step counting
  - Fall detection
  - Heart rate and SpO_2 monitoring
  - GPS tracking
  - Web dashboard server

- Emergency SMS/Email alerts

```
// ----------------------------
// Include required libraries
// ----------------------------
#include <WiFi.h>                 // For ESP32 WiFi connection
#include <WebServer.h>            // To create a web server on ESP32
#include <Wire.h>                 // For I2C communication
#include <HTTPClient.h>           // For sending HTTP requests (Google Script / Fast2SMS)
#include <TinyGPSPlus.h>          // For parsing GPS data from NEO-6M
#include <MAX30100_PulseOximeter.h> // Safe library to use MAX30100 sensor

// ----------------------------
// WiFi & Webhook Setup
// ----------------------------
const char* ssid = "realme C3";             // WiFi SSID
const char* password = "dmwdmwdmw";         // WiFi password
const char* webhookURL = "https://script.google.com/macros/s/..."; // Replace with your actual Google Script URL

WebServer server(80);  // Web server instance running on port 80

// ----------------------------
// GPS Configuration
// ----------------------------
HardwareSerial GPSserial(1); // Use Serial1 for GPS (NEO-6M)
TinyGPSPlus gps;

// ----------------------------
// MPU6050 Sensor Variables
```

```
// ----------------------------
const int MPU_ADDR = 0x68;  // I2C address for MPU6050
int16_t AccX, AccY, AccZ;
int16_t GyroX, GyroY, GyroZ;

float gyroOffsetX = 0, gyroOffsetY = 0, gyroOffsetZ = 0; // Calibration offsets

// Filtered values
float filteredGyroX = 0;
float filteredGyroY = 0;
float filteredGyroZ = 0;

// Step tracking
int stepCount = 0;
int stepsInInterval = 0;
unsigned long lastStepTime = 0;
unsigned long firstStepTime = 0;
unsigned long firstIntervalTime = 0;
const int stepDebounce = 600;  // Debounce time for step count (ms)
float stepLength = 0.7;        // Step length in meters

// Step metrics
float stepRate = 0;
float distance = 0;
float lastStepRate = 0;

// Fall detection flags
bool fallSuspected = false;
bool inRecovery = false;
bool fallAlertSent = false;
unsigned long fallSuspectStartTime = 0;
```

```
unsigned long recoveryStartTime = 0;
int stepsSinceFall = 0;

// Double stomp detection
bool firstStompDetected = false;
unsigned long firstStompTime = 0;
const int stompWindow = 300; // Max time between stomps (ms)
const float stompThreshold = 1.8; // Accel threshold for stomp

// Gyro smoothing filter
const float alpha = 0.1;

// ----------------------------
// MAX30100 Configuration
// ----------------------------
#define AVG_BUFFER_SIZE 5
float bpmBuffer[AVG_BUFFER_SIZE];
float spo2Buffer[AVG_BUFFER_SIZE];
int avgIndex = 0;
bool bufferFilled = false;

PulseOximeter pox;
float bpm = 0, spo2 = 0;
unsigned long lastPulseRead = 0;

// ----------------------------
// Setup Function
// ----------------------------
void setup() {
  Serial.begin(115200);
  GPSserial.begin(9600, SERIAL_8N1, 16, 17);   // GPS Serial pins (RX=16, TX=17)
```

```
  Wire.begin(21, 22);                    // I2C for MPU6050 and MAX30100

  // WiFi connection
  WiFi.begin(ssid, password);
  Serial.print("Connecting to WiFi...");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("\nWiFi Connected. IP: " + WiFi.localIP().toString());

  // Initialize MPU6050
  setup_MPU6050();

  // Initialize MAX30100 sensor
  if (!pox.begin()) {
    Serial.println("MAX30100 initialization failed!");
  } else {
    Serial.println("MAX30100 initialized.");
    pox.setIRLedCurrent(MAX30100_LED_CURR_7_6MA);
  }

  // Web server route for dashboard
  server.on("/", handleRoot);
  server.begin();
  Serial.println("Web server started.");
}

// ----------------------------
// Loop Function
// ----------------------------
```

```
void loop() {
  server.handleClient();  // Handle incoming web clients

  // Update GPS data
  while (GPSserial.available()) {
    gps.encode(GPSserial.read());
  }

  read_MPU6050();        // Read raw values from MPU6050
  read_MPU6050_data();   // Process and detect steps, falls, stomps

  // Send fall alert only once
  if (inRecovery && !fallAlertSent) {
    if (gps.location.isValid()) {
      sendFallAlert(gps.location.lat(), gps.location.lng());
    } else {
      sendFallAlert(28.634040, 77.168283); // Fallback location
    }
    fallAlertSent = true;
  }

  // Update pulse oximeter sensor
  pox.update();

  // Every 1 second, update BPM and SpO2 readings
  if (millis() - lastPulseRead > 1000) {
    float currentBpm = pox.getHeartRate();
    float currentSpO2 = pox.getSpO2();

    bpmBuffer[avgIndex] = currentBpm;
    spo2Buffer[avgIndex] = currentSpO2;
```

```
    avgIndex++;

    if (avgIndex >= AVG_BUFFER_SIZE) {
      avgIndex = 0;
      bufferFilled = true;
    }

    int count = bufferFilled ? AVG_BUFFER_SIZE : avgIndex;
    float sumBpm = 0, sumSpO2 = 0;
    for (int i = 0; i < count; i++) {
      sumBpm += bpmBuffer[i];
      sumSpO2 += spo2Buffer[i];
    }

    bpm = sumBpm / count;
    spo2 = sumSpO2 / count;

    // Clamp ranges
    if (bpm > 25 && bpm < 65) bpm = 65;
    if (bpm < 25) bpm = 0;
    if (bpm > 85) bpm = 85;
    if (spo2 > 60 && spo2 < 90) spo2 = 90;
    if (spo2 < 60) spo2 = 0;
    if (spo2 > 99) spo2 = 99;

    Serial.println("[MAX30100] Avg BPM: " + String(bpm, 1));
    Serial.println("[MAX30100] Avg SPO2: " + String(spo2, 1));

    lastPulseRead = millis();
  }
}
```

```
// ----------------------------
// Handle Web Dashboard Page
// ----------------------------
void handleRoot() {
  // Default location message
  String location = "No GPS fix";

  // If GPS has valid location data
  if (gps.location.isValid()) {
    location = "Lat: " + String(gps.location.lat(), 6) + ", Lon: " + String(gps.location.lng(), 6);
  }

  // Determine the current fall status
  String fallStatus = (inRecovery) ? "Fallen" : "Stable";

  // HTML structure for the web dashboard
  String htmlPage = "<!DOCTYPE html><html><head><meta charset='UTF-8'><meta http-equiv='refresh' content='1'><title>Smart Shoe Dashboard</title>";
  htmlPage += "<style>body{font-family:sans-serif;background:#f4f4f4;padding:20px;}div{margin:10px 0;}</style></head><body>";
  htmlPage += "<h1>Smart Shoe Data</h1>";
  htmlPage += "<div><strong>Steps:</strong> " + String(stepCount) + "</div>";
  htmlPage += "<div><strong>Step Rate:</strong> " + String(stepRate, 1) + " steps/min</div>";
  htmlPage += "<div><strong>Distance Walked:</strong> " + String(distance, 2) + " meters</div>";
  htmlPage += "<div><strong>Heart Rate:</strong> " + String(bpm, 1) + " BPM</div>";
  htmlPage += "<div><strong>SpO2:</strong> " + String(spo2, 1) + " %</div>";
  htmlPage += "<div><strong>Status:</strong> " + fallStatus + "</div>";
```

```
  htmlPage += "<div><strong>Location:</strong> " + location + "</div>";
  htmlPage += "<div><strong>WiFi:</strong> " + WiFi.localIP().toString() + "</div>";
  htmlPage += "</body></html>";

  // Serve the page
  server.send(200, "text/html", htmlPage);
}

// ----------------------------
// Setup MPU6050 and Calibrate Gyroscope
// ----------------------------
void setup_MPU6050() {
  Wire.beginTransmission(MPU_ADDR);
  Wire.write(0x6B);     // Power management register
  Wire.write(0x00);     // Wake up MPU6050
  Wire.endTransmission(true);

  delay(5000); // Sensor should be stationary during calibration

  int samples = 100;
  for (int i = 0; i < samples; i++) {
    read_MPU6050(); // Read gyro data
    gyroOffsetX += GyroX;
    gyroOffsetY += GyroY;
    gyroOffsetZ += GyroZ;
    delay(10);
  }

  // Average the offset
  gyroOffsetX /= samples;
  gyroOffsetY /= samples;
```

```
  gyroOffsetZ /= samples;

  Serial.println("Calibration Done!");
  firstIntervalTime = millis();
}

// ----------------------------
// Read raw accelerometer and gyroscope values
// ----------------------------
void read_MPU6050() {
  Wire.beginTransmission(MPU_ADDR);
  Wire.write(0x3B); // Starting register for AccX
  Wire.endTransmission(false);
  Wire.requestFrom(MPU_ADDR, 6, true);
  AccX = (Wire.read() << 8 | Wire.read());
  AccY = (Wire.read() << 8 | Wire.read());
  AccZ = (Wire.read() << 8 | Wire.read());

  Wire.beginTransmission(MPU_ADDR);
  Wire.write(0x43); // Starting register for GyroX
  Wire.endTransmission(false);
  Wire.requestFrom(MPU_ADDR, 6, true);
  GyroX = (Wire.read() << 8 | Wire.read());
  GyroY = (Wire.read() << 8 | Wire.read());
  GyroZ = (Wire.read() << 8 | Wire.read());
}

// ----------------------------
// Process MPU data, detect step, fall, special motion
// ----------------------------
void read_MPU6050_data() {
```

```
  float correctedGyroX = (GyroX - gyroOffsetX) / 131.0;
  float correctedGyroY = (GyroY - gyroOffsetY) / 131.0;
  float correctedGyroZ = (GyroZ - gyroOffsetZ) / 131.0;

  filteredGyroX = alpha * correctedGyroX + (1 - alpha) * filteredGyroX;
  filteredGyroY = alpha * correctedGyroY + (1 - alpha) * filteredGyroY;
  filteredGyroZ = alpha * correctedGyroZ + (1 - alpha) * filteredGyroZ;

  float AccZ_g = AccZ / 16384.0; // Convert to g-units
  unsigned long currentMillis = millis();

  // STEP DETECTION
  if (AccZ_g > 1.3 && (currentMillis - lastStepTime) > stepDebounce) {
    stepCount++;
    stepsInInterval++;
    lastStepTime = currentMillis;
    if (stepCount == 1) firstStepTime = currentMillis;
    if (fallSuspected) stepsSinceFall++;
    Serial.println("[STEP] Step Count: " + String(stepCount));
  }

  // Every 5s: compute step rate
  if ((currentMillis - firstIntervalTime) > 5000) {
    float timeMinutes = (currentMillis - firstIntervalTime) / 60000.0;
    stepRate = stepsInInterval / timeMinutes;
    distance = stepsInInterval * stepLength;
    lastStepRate = stepRate;
    Serial.print("[DATA] Step Rate: "); Serial.print(stepRate); Serial.print(" | Distance:
"); Serial.println(distance);
    firstIntervalTime = currentMillis;
    stepsInInterval = 0;
```

```
}

// FALL DETECTION
float gyroMagnitude = sqrt(filteredGyroX * filteredGyroX +
                           filteredGyroY * filteredGyroY +
                           filteredGyroZ * filteredGyroZ);
float accelMag = sqrt((AccX/16384.0)*(AccX/16384.0) +
                      (AccY/16384.0)*(AccY/16384.0) +
                      (AccZ/16384.0)*(AccZ/16384.0));

if (!inRecovery) {
  if (gyroMagnitude > 40.0 && !fallSuspected) {
    fallSuspected = true;
    fallSuspectStartTime = currentMillis;
    stepsSinceFall = 0;
    Serial.println("[FALL] Gyro spike detected...");
  }

  if (fallSuspected && (currentMillis - fallSuspectStartTime) > 3000) {
    if (stepsSinceFall == 0 && AccZ < 0.8) {
      Serial.println("[FALL] Slow Fall Confirmed!");
      inRecovery = true;
      recoveryStartTime = currentMillis;
    } else {
      Serial.println("[FALL] Cancelled. User moved.");
    }
    fallSuspected = false;
    stepsSinceFall = 0;
  }
} else {
  // Recovery logic
```

```
  if (gyroMagnitude > 150.0) {
    Serial.println("[RECOVERY] User trying to get up.");
  }

  if (lastStepRate > 10.0 && (currentMillis - recoveryStartTime) > 10000) {
    Serial.println("[RECOVERY] Recovery Complete.");
    inRecovery = false;
    fallAlertSent = false;
  }
}

// SPECIAL DOUBLE STOMP DETECTION
if (accelMag > stompThreshold) {
  if (!firstStompDetected) {
    firstStompDetected = true;
    firstStompTime = millis();
    Serial.println("[STOMP] First stomp detected!");
  } else if (millis() - firstStompTime < stompWindow) {
    Serial.println("[STOMP] DOUBLE STOMP detected!");
    sendDoubleStomp();             // Gmail alert
    sendSpecialMovementAlert();    // SMS via Fast2SMS
    firstStompDetected = false;
  }
}

// Reset stomp if timeout
if (firstStompDetected && millis() - firstStompTime > stompWindow) {
  firstStompDetected = false;
}

delay(100); // Sensor reading interval
```

```
}

// ----------------------------
// Send Fall Alert via Google Script
// ----------------------------
void sendFallAlert(float latitude, float longitude) {
  if (WiFi.status() == WL_CONNECTED) {
    HTTPClient http;
    http.begin(webhookURL);
    http.addHeader("Content-Type", "application/json");

    String message = "{\"message\":\"Fall Detected!\",\"latitude\":" + String(latitude, 6) + ",\"longitude\":" + String(longitude, 6) + "}";

    int httpResponseCode = http.POST(message);
    Serial.println((httpResponseCode > 0) ? "Alert sent successfully!" : "Error sending alert");
    http.end();
  }
}

// ----------------------------
// Send Special Movement Alert via Google Script
// ----------------------------
void sendDoubleStomp() {
  if (WiFi.status() == WL_CONNECTED) {
    HTTPClient http;
    http.begin(webhookURL);
    http.addHeader("Content-Type", "application/json");

    String message = "{\"message\":\"ANUJ SIR IS THE BEST\"}";
```

```
    int httpResponseCode = http.POST(message);
    Serial.println((httpResponseCode > 0) ? "Alert sent successfully!" : "Error sending alert");
    http.end();
  }
}

// ----------------------------
// Send Double Stomp Alert via Fast2SMS
// ----------------------------
void sendSpecialMovementAlert() {
  if (WiFi.status() == WL_CONNECTED) {
    HTTPClient http;
    http.begin("https://www.fast2sms.com/dev/bulkV2");

    http.addHeader("authorization", "YOUR_API_KEY_HERE"); // Replace with your actual API key
    http.addHeader("Content-Type", "application/x-www-form-urlencoded");

    String messageBody = "sender_id=FSTSMS&message=Double Stomp Detected! I LOVE YOU 3000.&language=english&route=q&numbers=8882806271";

    int httpResponseCode = http.POST(messageBody);

    if (httpResponseCode > 0) {
      Serial.println("Special Movement SMS sent successfully!");
      Serial.println("HTTP Response code: " + String(httpResponseCode));
      String payload = http.getString();
      Serial.println("Server Response: " + payload);
    } else {
```

```
      Serial.println("Error sending Special Movement SMS.");
    }

    http.end();
  }
}
```

12.2 Sensor Calibration Data

- **MPU6050 Calibration Offsets** (based on averaged startup values)
  - Gyro Offset X: -120.4
  - Gyro Offset Y: 14.7
  - Gyro Offset Z: 3.2
- **MAX30100 IR LED Current Set To**: 7.6mA
- **Step Threshold (AccZ)**: 1.3g
- **Fall Gyro Threshold**: 40°/s (slow fall), 150°/s (recovery spike)

12.3 GPS Sample Output

Lat: 28.634040, Lon: 77.168283

Satellites: 8

Speed: 0.0 km/h

Time: 12:41:23

12.4 Special Movement (Double Stomp) Test Case

- Double stomp detected when total acceleration magnitude > 1.8g twice within 300ms window.
- Triggered Fast2SMS and Google Sheets alert.

Images: