ElderGuard

Group Members:

Name	Student ID	E-mail
1. Amitaash Rao R	IMT2024060	AmitaashRao.R@iiitb.ac.in
2. Gaurav Ravindra Nayak	IMT2024077	Gaurav.Nayak@iiitb.ac.in
3. Harshvardhan Mishra	BT2024251	Harshvardhan.Mishra@iiitb.ac.in
4. Nirmeet Rajesh Udeshi	IMT2024043	NirmeetRajesh.Udeshi@iiitb.ac.in
5. Prashant VSG	IMT2024075	Prashant.Garimella@iiitb.ac.in
6. Rohan Kamath	BT2024238	Rohan.Kamath@iiitb.ac.in

ECE 211P: Electronics Lab
Prof. Madhav Rao

November 19, 2025

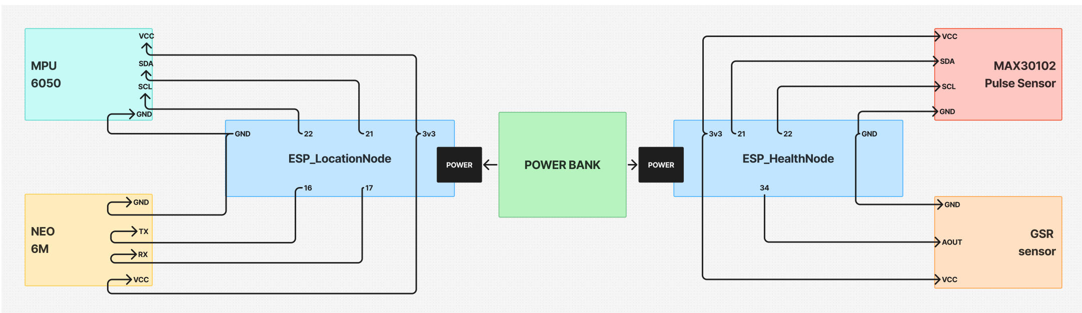

Figure 1: System Block Diagram: ESP32 Nodes and Cloud Architecture

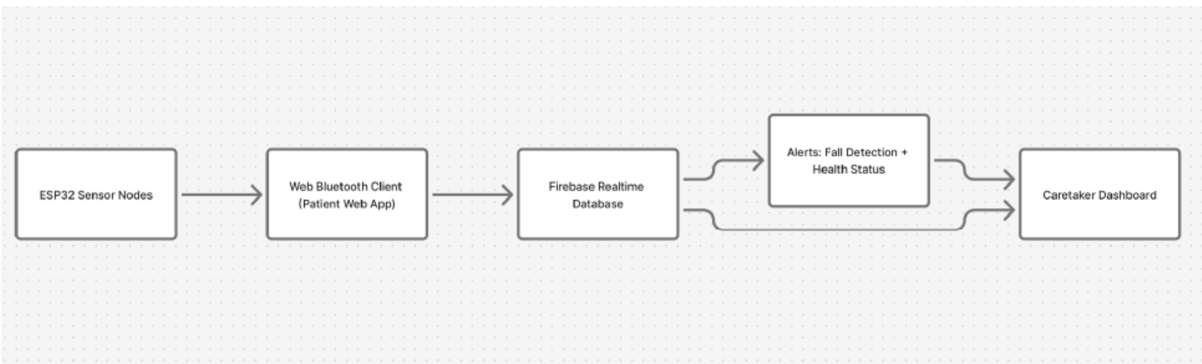

Figure 2: End-to-End Data Flow Architecture: ESP32 Sensor Nodes send real-time sensor data to the Patient Web App via Web Bluetooth.

Report by Amitaash Rao R

Contents

1 Abstract

This report presents the design and implementation of a smart jacket for elderly individuals, aimed at enhancing personal safety and health monitoring. The system integrates multiple sensors to detect falls, track location, monitor heart rate via photoplethysmography (PPG), and assess panic states through galvanic skin response (GSR). Utilizing two ESP32 microcontrollers for distributed processing, data is transmitted via Bluetooth Low Energy (BLE) to a custom web application. This application acts as a gateway, pushing real-time data to Google Firebase, which synchronizes with a remote caretaker dashboard for live monitoring and emergency alerts.

2 Introduction

Falls among the elderly represent a significant public health concern. Traditional monitoring methods often lack the precision needed for proactive intervention. This project addresses these challenges by developing a smart jacket embedded with advanced sensors to provide real-time detection and response capabilities. The system identifies falls using inertial data and continuously monitors physiological parameters to predict risk states. A crucial component of this system is its cloud connectivity, ensuring that data is not just collected locally but is instantly available to caregivers anywhere in the world.

3 Theory and Implementation

The system is divided into three layers: The Sensing Layer (ESP32 Nodes), the Gateway Layer (Patient Web App), and the Monitoring Layer (Caretaker Dashboard).

3.1 Motion Processing (ESP32-A)

The ESP32-A polls the MPU6050 IMU to detect falls. The logic processes raw sensor data every 50ms (20 Hz). The raw accelerometer (AcX, AcY, AcZ) and gyroscope (GyX, GyY, GyZ) readings are normalized using sensitivity factors $S_a = 16384.0$ (for $\pm 2g$) and $S_g = 131.07$ (for $\pm 250°/s$).

Total Acceleration Amplitude (Amp):

$$Amp = 10 \cdot \sqrt{\left(\frac{AcX}{S_a}\right)^2 + \left(\frac{AcY}{S_a}\right)^2 + \left(\frac{AcZ}{S_a}\right)^2}$$

Where Amp is scaled by 10 for integer threshold comparison. A value of $Amp \approx 10$ represents $1g$ (static gravity).

Angular Velocity Magnitude ($\Delta\phi$):

$$\Delta\phi = \sqrt{\left(\frac{GyX}{S_g}\right)^2 + \left(\frac{GyY}{S_g}\right)^2 + \left(\frac{GyZ}{S_g}\right)^2}$$

Where $\Delta\phi$ represents the total rate of rotation in degrees per second.

Fall Triggers:

1. Freefall: $Amp < 3$ (approx $0.3g$).
2. Impact: $Amp > 5$ (approx $0.5g$) shortly after freefall.
3. Orientation: $\Delta\phi > 20$ shortly after impact.

Data Rates: GPS data is transmitted via BLE every 5000ms (0.2 Hz). Fall alerts are asynchronous (immediate).

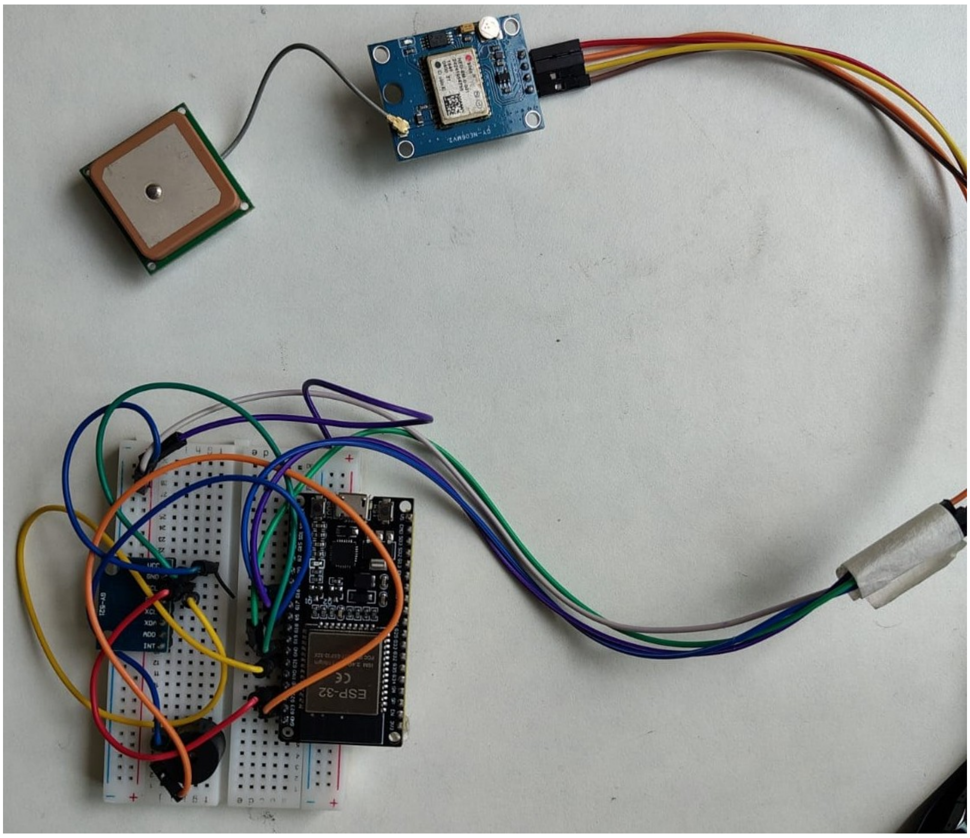

Figure 3: Hardware implementation of the Motion Node: ESP32 interfaced with the MPU6050 IMU and NEO-6M GPS module.

3.2 Health Monitor Sensing (ESP32-B)

The ESP32-B processes health metrics. The MAX30105 sensor is configured to sample at 400 Hz to capture precise pulse peaks.

Galvanic Skin Response (GSR): Skin resistance (R_{skin}) is calculated using a voltage divider with a fixed resistor $R_1 = 10k\Omega$ and supply voltage $V_{CC} = 3.3V$. The ADC reading (raw) is averaged over 10 samples.

$$V_{out} = \frac{raw}{4095.0} \cdot V_{CC}$$

$$R_{skin} = R_1 \cdot \left(\frac{V_{CC} - V_{out}}{V_{out}} \right)$$

$R_{skin} < 40k\Omega$ indicates high stress/panic.

Heart Rate (BPM): Calculated by measuring the time delta (Δt) in milliseconds between valid IR signal peaks.

$$BPM = \frac{60}{\Delta t/1000.0}$$

Values are averaged over a 3-beat window. $BPM < 50$ indicates bradycardia/fainting risk.

Data Rates: The aggregated health packet (JSON) is notified via BLE every 500ms (2 Hz).

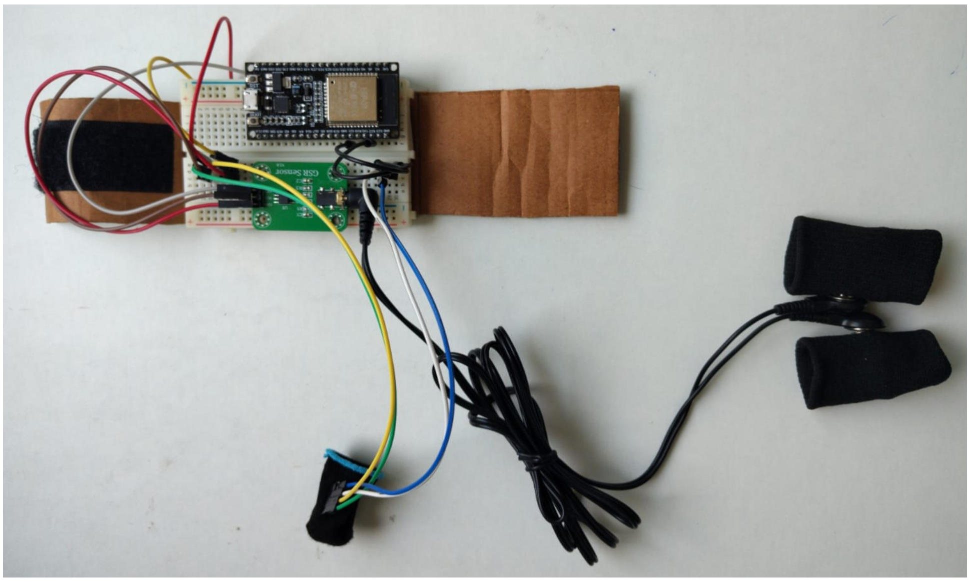

Figure 4: Hardware implementation of the Health Node: ESP32 interfaced with the MAX30102 Pulse Oximeter and GSR sensing circuit.

4 Apparatus

- **Microcontrollers:** 2 × ESP32-WROOM-32.
- **Sensors:** MPU6050 (IMU), NEO-6M (GPS), MAX30105 (PPG), Analog GSR.
- **Software Stack:** Firmware (C++), Frontend (HTML5/JS Web Bluetooth), Backend (Firebase Realtime DB).
- **Power:** 5V USB Power Bank.

Figure 5: Complete Wearable Hardware Assembly: Smart jacket incorporating the ESP32 motion-detection node, GPS module, health-monitoring node, GSR electrodes, and wiring harness.

5 Web Application & Cloud Architecture

The connectivity architecture bridges the physical wearable with the digital cloud.

5.1 Sender Application (Patient Side)

A web-based application runs on the patient's smartphone. It utilizes the **Web Bluetooth API** to scan for and connect to the ESP32 nodes. Upon receiving JSON packets via BLE notifications, the app stamps the data with a precise timestamp and uploads it to the Firebase Realtime Database using the REST API.

5.2 Caretaker Dashboard (Remote Side)

The dashboard listens for database changes in real-time.

- **Live Vitals:** Updates heart rate and stress status instantly.
- **Location Tracking:** Embeds Google Maps links for the last known coordinates.
- **Emergency Alerts:** If a "fall_detected" flag is received, the dashboard triggers an intrusive modal popup and plays an audible alarm.

6 Results

The system achieved full end-to-end connectivity. The ESP32 nodes successfully transmitted JSON data to the mobile web app, which synced with Firebase in under 200ms.

Event	Action	Cloud Latency
Routine Heartbeat	Update Dashboard	$\approx$ 150ms
Fall Detected	Trigger Audio Alarm	$\approx$ 300ms

Table 1: System Latency Performance

The Caretaker Dashboard correctly rendered the Google Maps location upon a fall event, validating the GPS integration.

7 Project Demonstration

A comprehensive video demonstration of the ElderGuard system in operation, showing the fall detection mechanism, real-time health monitoring updates, and the cloud-connected caretaker dashboard response, can be accessed via the following Google Drive link:

Video Link:

https://drive.google.com/file/d/1ESebDfQ9aW7mKUBl7Qa6kqjXh2BnNB3O/view?usp=sharing

8 Conclusion

This project successfully demonstrated a complete IoT solution for elderly care. By combining edge computing (on-device fall detection) with cloud connectivity (Firebase), the system offers robust safety features without compromising on battery life or usability. The use of standard web technologies makes the solution highly accessible and scalable.

9 References

References

[1] Espressif Systems, "ESP32 Series Datasheet," ver. 4.0, 2022.

[2] InvenSense, "MPU-6000 and MPU-6050 Product Specification," ver. 3.4, 2013.

[3] Maxim Integrated, "MAX30105: High-Sensitivity Pulse Oximeter," Rev 1, 2018.

[4] F. Saleh, "Fall detection using inertial sensors," *IEEE Sensors Journal*, 2020.

[5] MDN, "Web Bluetooth API Specification," 2023.

[6] Google, "Firebase Realtime Database Documentation," 2023.

Appendices

Appendix A: ESP32-A Firmware (Motion & Location)

```
#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>
#include <Wire.h>
#include <MPU6050.h>
#include <TinyGPSPlus.h>

BLEServer* pServer = NULL;
BLECharacteristic* pCharacteristic = NULL;
MPU6050 mpu;
TinyGPSPlus gps;
#define gpsSerial Serial2

bool deviceConnected = false;
int buzPin = 5;
unsigned long lastGPSPrint = 0;

#define SERVICE_UUID        "6e400001-b5a3-f393-e0a9-e50e24dcca9e"
#define CHARACTERISTIC_UUID "6e400003-b5a3-f393-e0a9-e50e24dcca9e"

int16_t AcX, AcY, AcZ, GyX, GyY, GyZ;
float ax, ay, az, gx, gy, gz;
boolean trigger1=false, trigger2=false, trigger3=false;
byte trigger1count=0, trigger2count=0;
boolean fall = false;
float angleChange = 0;

#define FREEFALL_THRESHOLD    3
#define IMPACT_THRESHOLD      5
#define ORIENTATION_THRESHOLD 20
```

```

class MyServerCallbacks : public BLEServerCallbacks {
  void onConnect(BLEServer* pServer) { deviceConnected = true; }
  void onDisconnect(BLEServer* pServer) {
    deviceConnected = false;
    pServer->startAdvertising();
  }
};

void setup() {
  Serial.begin(115200);
  gpsSerial.begin(9600, SERIAL_8N1, 16, 17);
  Wire.begin(21, 22);
  mpu.initialize();

  BLEDevice::init("ESP32_LocationNode");
  pServer = BLEDevice::createServer();
  pServer->setCallbacks(new MyServerCallbacks());
  BLEService *pService = pServer->createService(SERVICE_UUID);
  pCharacteristic = pService->createCharacteristic(CHARACTERISTIC_UUID,
    BLECharacteristic::PROPERTY_NOTIFY);
  pCharacteristic->addDescriptor(new BLE2902());
  pService->start();
  BLEDevice::startAdvertising();

  pinMode(buzPin, OUTPUT);
}

void loop() {
  while (gpsSerial.available() > 0) gps.encode(gpsSerial.read());
  mpu.getMotion6(&AcX, &AcY, &AcZ, &GyX, &GyY, &GyZ);

  ax = (AcX-2050)/16384.0; ay = (AcY-77)/16384.0; az = (AcZ-1947)
   /16384.0;
  gx = (GyX+270)/131.07; gy = (GyY-351)/131.07; gz = (GyZ+136)/131.07;
```

```
float Amp = sqrt(ax*ax + ay*ay + az*az) * 10;
angleChange = sqrt(gx*gx + gy*gy + gz*gz);

if (Amp <= FREEFALL_THRESHOLD && !trigger2) trigger1 = true;

if (trigger1) {
  trigger1count++;
  if (Amp >= IMPACT_THRESHOLD) {
    trigger2 = true; trigger1 = false; trigger1count = 0;
  }
}

if (trigger2) {
  trigger2count++;
  if (angleChange >= ORIENTATION_THRESHOLD) {
    trigger3 = true; trigger2 = false; trigger2count = 0;
  }
}

bool lyingPosture = (abs(az) > 0.7) && (abs(ay) < 0.4);
if (trigger3 && lyingPosture) {
  fall = true; trigger3 = false;
  digitalWrite(buzPin, HIGH);

  float lat = gps.location.isValid() ? gps.location.lat() : 0.0;
  float lng = gps.location.isValid() ? gps.location.lng() : 0.0;
  String fallJson = "{\"fall_detected\":true,\"lat\":" + String(lat
 ,6) + ",\"lon\":" + String(lng,6) + "}";

  if (deviceConnected) {
    pCharacteristic->setValue(fallJson.c_str());
    pCharacteristic->notify();
  }
  delay(2000);
```

```
    digitalWrite(buzPin, LOW);
    lastGPSPrint = millis();
  }

  if (trigger1count >= 10) { trigger1 = false; trigger1count = 0; }
  if (trigger2count >= 10) { trigger2 = false; trigger2count = 0; }

  if (millis() - lastGPSPrint > 5000) {
    lastGPSPrint = millis();
    if (gps.location.isValid() && deviceConnected) {
      String gpsJson = "{\"lat\":" + String(gps.location.lat(), 6) + ",\"lon\":" + String(gps.location.lng(), 6) + "}";
      pCharacteristic->setValue(gpsJson.c_str());
      pCharacteristic->notify();
    }
  }
  delay(50);
}
```

Listing 1: Motion Node Firmware

Appendix B: ESP32-B Firmware (Health Monitoring)

```
#include <Wire.h>
#include "MAX30105.h"
#include "heartRate.h"
#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>

BLEServer* pServer = NULL;
BLECharacteristic* pCharacteristic = NULL;
#define SERVICE_UUID        "6e400001-b5a3-f393-e0a9-e50e24dcca9e"
#define CHARACTERISTIC_UUID "6e400003-b5a3-f393-e0a9-e50e24dcca9e"

#define GSR_PIN 34
#define VCC 3.3
#define ADC_MAX 4095
int rawValue = 0;
float voltage=0.0, resistance=0.0;
String state;

MAX30105 particleSensor;
const byte RATE_SIZE = 3;
byte rates[RATE_SIZE];
byte rateSpot = 0;
long lastBeat = 0;
float beatsPerMinute = 0;
int beatAvg = 0;
bool deviceConnected = false;

class MyServerCallbacks : public BLEServerCallbacks {
  void onConnect(BLEServer* pServer) { deviceConnected = true; }
  void onDisconnect(BLEServer* pServer) {
    deviceConnected = false;
    pServer->startAdvertising();
```

```
  }
};

void setup() {
  Serial.begin(115200);
  if (!particleSensor.begin(Wire, I2C_SPEED_FAST)) while(1);
  particleSensor.setup(0x1F, 4, 2, 400, 411, 4096);
  particleSensor.setPulseAmplitudeRed(0x15);
  particleSensor.setPulseAmplitudeGreen(0x00);
  pinMode(GSR_PIN, INPUT);

  BLEDevice::init("ESP32_HealthNode");
  pServer = BLEDevice::createServer();
  pServer->setCallbacks(new MyServerCallbacks());
  BLEService *pService = pServer->createService(SERVICE_UUID);
  pCharacteristic = pService->createCharacteristic(CHARACTERISTIC_UUID,
    BLECharacteristic::PROPERTY_NOTIFY);
  pCharacteristic->addDescriptor(new BLE2902());
  pService->start();
  BLEDevice::startAdvertising();
}

void loop() {
  long sum = 0;
  for (int i=0; i<10; i++) { sum += analogRead(GSR_PIN); delay(3); }
  rawValue = sum / 10;
  voltage = (rawValue / (float)ADC_MAX) * VCC;
  resistance = (10.0 * (VCC - voltage)) / voltage;

  if (resistance > 80) state = "Relaxed";
  else if (resistance > 40) state = "Calm";
  else if (resistance > 20) state = "Stressed";
  else state = "Very Stressed";

  long irValue = particleSensor.getIR();
```

```
  if (irValue > 50000 && checkForBeat(irValue)) {
    long delta = millis() - lastBeat;
    lastBeat = millis();
    beatsPerMinute = 60 / (delta / 1000.0);
    if (beatsPerMinute < 200 && beatsPerMinute > 35) {
      rates[rateSpot++] = (byte)beatsPerMinute;
      rateSpot %= RATE_SIZE;
      beatAvg = 0;
      for (byte x=0; x<RATE_SIZE; x++) beatAvg += rates[x];
      beatAvg /= RATE_SIZE;
    }
  }

  String fallRisk = (beatAvg < 50 || resistance < 40) ? "Yes" : "No";
  String jsonData = "{\"status\":\"" + state + "\",\"avg_bpm\":" +
   String(beatAvg) + ",\"fall_risk\":\"" + fallRisk + "\"}";

  static unsigned long lastPrint = 0;
  if (millis() - lastPrint > 500) {
    lastPrint = millis();
    if (deviceConnected) {
      pCharacteristic->setValue(jsonData.c_str());
      pCharacteristic->notify();
    }
  }
  delay(10);
}
```

Listing 2: Health Node Firmware

Appendix C: Web Application Logic (JavaScript)

```javascript
async function connectToBLE(deviceConfig) {
  const bleDevice = await navigator.bluetooth.requestDevice({
    filters: [{ namePrefix: deviceConfig.namePrefix }],
    optionalServices: [UART_SERVICE_UUID],
  });
  const server = await bleDevice.gatt.connect();
  const service = await server.getPrimaryService(UART_SERVICE_UUID);
  const tx = await service.getCharacteristic(CHAR_UUID_TX);
  await tx.startNotifications();

  tx.addEventListener("characteristicvaluechanged", (event) => {
    const dataString = new TextDecoder().decode(event.target.value);
    const jsonData = JSON.parse(dataString);
    jsonData.timestamp = new Date().toISOString();
    postToFirebase(deviceConfig.firebaseUrl, jsonData);
  });
}

async function postToFirebase(url, data) {
  await fetch(url, {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify(data),
  });
}
```

Listing 3: Sender App (Connectivity & Upload)

```javascript
db.ref('/fall_alerts').limitToLast(1).on('value', (snap) => {
  if (!snap.exists()) return;
  const data = Object.values(snap.val())[0];
  const fallPopup = document.getElementById('fallAlertPopup');
  fallPopup.style.display = 'block';
  fallPopup.textContent = "[ALERT] Fall Detected at " + formatTime(data
    .timestamp);
```

```
  const audio = new Audio('https://actions.google.com/sounds/v1/alarms/beep_short.ogg');
  audio.play();
});
```

Listing 4: Receiver Dashboard (Alert Logic)