LAMPIRAN

Lampiran 1. Hasil Turnitin

UNIVERSITAS DARMA PERSADA
UPT PERPUSTAKAAN
Gedung Rektorat Lantai 3,
Jl.Taman Malaka Selatan, Pondok Kelapa – Jakarta Timur 13450

SURAT KETERANGAN
HASIL PENGECEKAN TURNITIN

UPT Perpustakaan Universitas Darma Persada menerangkan telah selesai melakukan pemeriksaan duplikasi/*similarity* menggunakan perangkat lunak Turnitin terhadap hasil karya sebagai berikut:

Judul : Alat Perlindungan dan Deteksi Gerakan Jatuh Pada Lansia Berbasis IoT

Penulis : Wisnu Anggoro Laras

NIM : **2021230038**

Tgl pemeriksaan : 24 Juli 2025

Dengan hasil Tingkat Kesamaan (*similarity index*) **7%**

Demikian Surat Keterangan kami buat, untuk dipergunakan sebagaimana mestinya.

Jakarta, 22 Juli 2025

Ka.UPT Perpustakaan Unsada

Yus Rusmiyati, SS., MM

Batas maksimal similarity 30% untuk Fakultas Sastra dan Ekonomi

Batas maksimal similarity 25% untuk Fakultas Teknik, Kelautan dan Pasca Sarjana

Lampiran 2. Source Code Arduino

```
#include <Wire.h>
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
#include <WiFi.h>
#include <time.h>
#include <HTTPClient.h>
#include <ArduinoJson.h>
#include <TinyGPS++.h>
#include <HardwareSerial.h>

Adafruit_MPU6050 mpu;
TinyGPSPlus gps;

// Define GPS module connection pins
#define GPS_RX_PIN 13
#define GPS_TX_PIN 12
HardwareSerial GPSSerial(1);

// Define Ultrasonic sensor pins
#define TRIG_PIN 14
#define ECHO_PIN 27

float distance = 0;
const float PROXIMITY_THRESHOLD = 30.0;

// WiFi Configuration - UBAH SESUAI JARINGAN ANDA
#define WIFI_SSID "AndromedaOS"      // Ganti dengan SSID WiFi Anda
#define WIFI_PASSWORD "Andromeda!@#" // Ganti dengan password WiFi Anda

// Define buzzer pin
#define BUZZER_PIN 5
#define GPS_BAUD 9600
unsigned long lastGPSDebugTime = 0;
const unsigned long GPS_DEBUG_INTERVAL = 5000;

// Device identification
const char* deviceID = "wheelchair_sensor_01";

// Firebase Configuration - GUNAKAN FIREBASE REST API
const char* FIREBASE_HOST = "test-firebase-message-4b729-default-rtdb.asia-southeast1.firebasedatabase.app";

const char* FIREBASE_AUTH = ""; // Kosongkan jika database rules allow public access
String FIREBASE_URL = "https://test-firebase-message-4b729-default-rtdb.asia-southeast1.firebasedatabase.app";

// FIRESTORE REST API - ALTERNATIF UNTUK REAL-TIME DATABASE
const char* FIRESTORE_PROJECT_ID = "test-firebase-message-4b729";
String FIRESTORE_BASE_URL = "https://firestore.googleapis.com/v1/projects/test-firebase-message-4b729
/databases/(default)/documents";

// IMPROVED: Wheelchair fall detection parameters - Lebih realistis
const float TILT_THRESHOLD = 25.0;              // Lebih rendah untuk wheelchair
const float IMPACT_THRESHOLD = 1.8;             // Lebih sensitif untuk impact
const int INACTIVITY_DURATION = 2000;           // 2 detik saja
const float ABNORMAL_ANGLE_THRESHOLD = 35.0;    // Lebih rendah untuk wheelchair
const float MOVEMENT_THRESHOLD = 1.2;           // Threshold untuk deteksi gerakan

// Buzzer alarm parameters
const int BUZZER_DURATION = 5000;
const int BUZZER_INTERVAL = 500;
```

```
unsigned long buzzerStartTime = 0;
bool buzzerActive = false;

// LED pins
#define LED_RED_PIN 16
#define LED_YELLOW_PIN 17
#define LED_GREEN_PIN 18

// GPS variables
float gpsLat = 0.0;
float gpsLng = 0.0;
bool validGPS = false;
unsigned long lastGPSUpdate = 0;
const unsigned long GPS_UPDATE_INTERVAL = 5000;

// For calculating orientation
float baselineAccelZ = 0;
float baselineAccelX = 0;
float baselineAccelY = 0;
const int CALIBRATION_SAMPLES = 100;

unsigned long fallDetectedTime = 0;
bool fallDetected = false;
bool calibrated = false;

// Ultrasonic sensor variables
unsigned long lastUltrasonicReading = 0;
const unsigned long ULTRASONIC_INTERVAL = 200;

// Status reporting interval
unsigned long lastStatusUpdate = 0;
const unsigned long STATUS_UPDATE_INTERVAL = 10000; // 10 seconds

// DEBUG: Enhanced debugging variables
unsigned long lastDebugOutput = 0;
const unsigned long DEBUG_INTERVAL = 3000; // Print debug every 3 seconds
bool debugMode = true;
int fallTestCounter = 0;
bool manualTestMode = false;

// IMPROVED: Movement detection variables
float lastAccelMagnitude = 0;
unsigned long lastMovementTime = 0;
int consecutiveTiltReadings = 0;
const int REQUIRED_CONSECUTIVE_TILTS = 2; // Lebih mudah trigger

// Fall detection state machine
enum FallState {
  MONITORING,
  TILT_DETECTED,
  IMPACT_DETECTED,
  INACTIVITY_MONITORING
};
```

```
FallState currentState = MONITORING;
String stateNames[] = {"MONITORING", "TILT_DETECTED", "IMPACT_DETECTED", "INACTIVITY_MONITORING"};

// DEBUG: Function to print with timestamp
void debugPrint(String message) {
  if (debugMode) {
    Serial.print("[");
    Serial.print(millis());
    Serial.print("ms] ");
    Serial.println(message);
  }
}

void setup() {
  Serial.begin(115200);
  while (!Serial) delay(10);

  debugPrint("=== WHEELCHAIR FALL DETECTION SYSTEM v2.0 ===");
  debugPrint("Starting system initialization...");

  // Initialize pins
  pinMode(TRIG_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);
  pinMode(BUZZER_PIN, OUTPUT);
  pinMode(LED_RED_PIN, OUTPUT);
  pinMode(LED_YELLOW_PIN, OUTPUT);
  pinMode(LED_GREEN_PIN, OUTPUT);

  digitalWrite(BUZZER_PIN, LOW);
  digitalWrite(LED_RED_PIN, LOW);
  digitalWrite(LED_YELLOW_PIN, LOW);
  digitalWrite(LED_GREEN_PIN, LOW);

  debugPrint("✅ GPIO pins initialized");

  // Initialize GPS
  setupGPS();

  // Test LEDs
  testLEDs();

  // Connect to WiFi
  connectToWiFi();

  // Initialize MPU6050
  if (!mpu.begin()) {
    debugPrint("❌ Failed to connect to MPU6050");
    sendStatusToFirestore("OFFLINE", "MPU6050 sensor connection failed", true);
    while (1) {
      digitalWrite(LED_RED_PIN, HIGH);
      delay(500);
      digitalWrite(LED_RED_PIN, LOW);
      delay(500);
    }
  }
  debugPrint("✅ MPU6050 connected successfully");
```

```
  // Configure sensor
  mpu.setAccelerometerRange(MPU6050_RANGE_8_G);
  mpu.setGyroRange(MPU6050_RANGE_500_DEG);
  mpu.setFilterBandwidth(MPU6050_BAND_21_HZ);

  // Test buzzer
  testBuzzer();

  // Calibration
  debugPrint("🔧 Starting calibration - keep wheelchair stationary...");
  delay(2000);
  calibrateSensor();

  debugPrint("✅ Calibration complete. System ready for fall detection!");
  debugPrint("🎯 FALL DETECTION THRESHOLDS:");
  debugPrint("   - Tilt Threshold: " + String(TILT_THRESHOLD) + "°");
  debugPrint("   - Impact Threshold: " + String(IMPACT_THRESHOLD) + " g");
  debugPrint("   - Abnormal Angle: " + String(ABNORMAL_ANGLE_THRESHOLD) + "°");
  debugPrint("   - Inactivity Duration: " + String(INACTIVITY_DURATION) + "ms");

  debugPrint("📋 SERIAL COMMANDS:");
  debugPrint("   - TEST_FALL: Trigger manual fall test");
  debugPrint("   - DEBUG_ON: Enable debug mode");
  debugPrint("   - DEBUG_OFF: Disable debug mode");
  debugPrint("   - STATUS: Show current system status");
  debugPrint("   - RESET: Reset system to monitoring");

  sendStatusToFirestore("monitoring", "System calibrated and active - v2.0", false);
}

void connectToWiFi() {
  debugPrint("📡 Connecting to WiFi: " + String(WIFI_SSID));
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);

  int attempts = 0;
  while (WiFi.status() != WL_CONNECTED && attempts < 30) {
    delay(500);
    Serial.print(".");
    attempts++;
  }
```

```
  if (WiFi.status() == WL_CONNECTED) {
    Serial.println();
    debugPrint("✅ WiFi connected!");
    debugPrint("📍 IP Address: " + WiFi.localIP().toString());
    debugPrint("📶 Signal Strength: " + String(WiFi.RSSI()) + " dBm");
    digitalWrite(LED_GREEN_PIN, HIGH);
    delay(1000);
    digitalWrite(LED_GREEN_PIN, LOW);
  } else {
    debugPrint("❌ WiFi connection failed! Running in offline mode.");
    digitalWrite(LED_RED_PIN, HIGH);
    delay(2000);
    digitalWrite(LED_RED_PIN, LOW);
  }
}

void testBuzzer() {
  debugPrint("🔊 Testing buzzer...");
  digitalWrite(BUZZER_PIN, HIGH);
  delay(200);
  digitalWrite(BUZZER_PIN, LOW);
  debugPrint("✅ Buzzer test complete");
}

void calibrateSensor() {
  float sumX = 0, sumY = 0, sumZ = 0;

  debugPrint("🔧 Collecting " + String(CALIBRATION_SAMPLES) + " calibration samples...");

  for (int i = 0; i < CALIBRATION_SAMPLES; i++) {
    sensors_event_t a, g, temp;
    mpu.getEvent(&a, &g, &temp);

    sumX += a.acceleration.x;
    sumY += a.acceleration.y;
    sumZ += a.acceleration.z;

    if (i % 20 == 0) {
      Serial.print(".");
    }
    delay(10);
  }
  Serial.println();

  baselineAccelX = sumX / CALIBRATION_SAMPLES;
  baselineAccelY = sumY / CALIBRATION_SAMPLES;
  baselineAccelZ = sumZ / CALIBRATION_SAMPLES;

  calibrated = true;
```

```
  debugPrint("✅ Baseline orientation calibrated:");
  debugPrint("   X: " + String(baselineAccelX) + " m/s²");
  debugPrint("   Y: " + String(baselineAccelY) + " m/s²");
  debugPrint("   Z: " + String(baselineAccelZ) + " m/s²");
}

float calculateTiltAngle(float x, float y, float z) {
  float dotProduct = x * baselineAccelX + y * baselineAccelY + z * baselineAccelZ;
  float magA = sqrt(x * x + y * y + z * z);
  float magB = sqrt(baselineAccelX * baselineAccelX + baselineAccelY
  * baselineAccelY + baselineAccelZ * baselineAccelZ);

  if (magA == 0 || magB == 0) return 0;

  float cosAngle = dotProduct / (magA * magB);
  cosAngle = constrain(cosAngle, -1.0, 1.0);

  return acos(cosAngle) * RAD_TO_DEG;
}

void updateGPS() {
  unsigned long startReadTime = millis();
  bool newData = false;

  while (GPSSerial.available() > 0 && millis() - startReadTime < 100) {
    char c = GPSSerial.read();
    if (gps.encode(c)) {
      newData = true;
    }
  }

  if (newData && gps.location.isValid()) {
    gpsLat = gps.location.lat();
    gpsLng = gps.location.lng();
    validGPS = true;
  }

  if (millis() - lastGPSDebugTime > GPS_DEBUG_INTERVAL) {
    printGPSDebugInfo();
    lastGPSDebugTime = millis();
  }
```

```
  if (millis() > 10000 && gps.charsProcessed() < 10) {
    debugPrint("⚠️ WARNING: No GPS data received! Check wiring.");
  }
}

float readUltrasonicDistance() {
  digitalWrite(TRIG_PIN, LOW);
  delayMicroseconds(2);

  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);

  long duration = pulseIn(ECHO_PIN, HIGH, 30000); // 30ms timeout
  if (duration == 0) return -1; // No echo received

  float distance = duration * 0.0343 / 2;
  return distance;
}

// DEBUG: Enhanced debug output function
void printSystemDebug() {
  if (!debugMode || millis() - lastDebugOutput < DEBUG_INTERVAL) return;

 sensors_event_t a, g, temp;
 mpu.getEvent(&a, &g, &temp);

 float tiltAngle = calculateTiltAngle(a.acceleration.x, a.acceleration.y, a.acceleration.z);
 float accel_magnitude = sqrt(a.acceleration.x * a.acceleration.x +
                              a.acceleration.y * a.acceleration.y +
                              a.acceleration.z * a.acceleration.z);
 float gyro_magnitude = sqrt(g.gyro.x * g.gyro.x + g.gyro.y * g.gyro.y + g.gyro.z * g.gyro.z);

 debugPrint("=== SYSTEM DEBUG STATUS ===");
 debugPrint("🔄 State: " + stateNames[currentState]);
 debugPrint("📐 Tilt Angle: " + String(tiltAngle, 2) + "°
 (Threshold: " + String(TILT_THRESHOLD) + "°)");
 debugPrint("⚡ Accel Magnitude: " + String(accel_magnitude, 2) + " m/s²
 (Impact Threshold: " + String(IMPACT_THRESHOLD) + ")");
 debugPrint("🌀 Gyro Magnitude: " + String(gyro_magnitude, 2) + " rad/s");
 debugPrint("📏 Distance: " + String(distance, 1) + " cm");
 debugPrint("📶 WiFi: " + String
 (WiFi.status() == WL_CONNECTED ? "Connected" : "Disconnected"));
 debugPrint("🛰️  GPS: " + String(validGPS ? "Valid" : "Invalid"));
 debugPrint("🔔 Consecutive Tilts: " + String(consecutiveTiltReadings));
 debugPrint("⏱️  Last Movement: " + String(millis() - lastMovementTime) + "ms ago");
 debugPrint("============================");
```

```
  lastDebugOutput = millis();
}

// DEBUG: Handle serial commands
void handleSerialCommands() {
  if (Serial.available()) {
    String command = Serial.readString();
    command.trim();
    command.toUpperCase();

    debugPrint("📨 Command received: " + command);

    if (command == "TEST_FALL") {
      debugPrint("🧪 MANUAL FALL TEST TRIGGERED!");
      manualTestMode = true;
      fallTestCounter++;

      sensors_event_t a, g, temp;
      mpu.getEvent(&a, &g, &temp);
      float tiltAngle = calculateTiltAngle(a.acceleration.x, a.acceleration.y, a.acceleration.z);


      // Force emergency
      debugPrint("🚨 Simulating fall detection...");
      sendFallDataToFirestore(75.0, a.acceleration.x, a.acceleration.y, a.acceleration.z, 5.0);
      sendStatusToFirestore("EMERGENCY", "Manual test fall #" + String(fallTestCounter), true);
      activateLocalAlarm();

      // Reset to monitoring after 5 seconds
      delay(5000);
      currentState = MONITORING;
      manualTestMode = false;
      debugPrint("✅ Manual test completed, returning to monitoring");

    } else if (command == "DEBUG_ON") {
      debugMode = true;
      debugPrint("✅ Debug mode enabled");

    } else if (command == "DEBUG_OFF") {
      debugMode = false;
      Serial.println("Debug mode disabled");


    } else if (command == "STATUS") {
      printSystemStatus();

    } else if (command == "RESET") {
      debugPrint("🔄 System reset to monitoring state");
      currentState = MONITORING;
      consecutiveTiltReadings = 0;
      buzzerActive = false;
      digitalWrite(BUZZER_PIN, LOW);
      sendStatusToFirestore("monitoring", "System reset via serial command", false);

    } else if (command == "WIFI_STATUS") {
      debugPrint("📡 WiFi Status: " + String(WiFi.status() == WL_CONNECTED ? "Connected" : "Disconnected"));
      if (WiFi.status() == WL_CONNECTED) {
        debugPrint("🔑 IP: " + WiFi.localIP().toString());
        debugPrint("📶 RSSI: " + String(WiFi.RSSI()) + " dBm");
      }
```

```
    } else if (command == "FIREBASE_TEST") {
      debugPrint("□ Testing Firebase connection...");
      sendStatusToFirestore("testing", "Firebase connection test via serial", false);

    } else {
      debugPrint("? Unknown command: " + command);
      debugPrint("Available commands: TEST_FALL, DEBUG_ON, DEBUG_OFF, STATUS, RESET, WIFI_STATUS, FIREBASE_TEST");
    }
  }
}

void printSystemStatus() {
  debugPrint("=== SYSTEM STATUS REPORT ===");
  debugPrint("Device ID: " + String(deviceID));
  debugPrint("Uptime: " + String(millis() / 1000) + " seconds");
  debugPrint("Current State: " + stateNames[currentState]);
  debugPrint("WiFi: " + String(WiFi.status() == WL_CONNECTED ? "Connected" : "Disconnected"));
  debugPrint("□ GPS: " + String(validGPS ? "Valid" : "Invalid"));
  debugPrint("Calibrated: " + String(calibrated ? "Yes" : "No"));
  debugPrint("□ Test Counter: " + String(fallTestCounter));
  debugPrint("============================");
}

void loop() {
  if (!calibrated) {
    calibrateSensor();
    return;
  }

  // Handle serial commands for debugging
  handleSerialCommands();

  // Check WiFi connection
  if (WiFi.status() != WL_CONNECTED) {
    debugPrint("⚠□ WiFi disconnected. Attempting to reconnect...");
    connectToWiFi();
  }

  updateGPS();
  updateBuzzer();

  // Print debug info periodically
  printSystemDebug();

  // Read ultrasonic sensor
  if (millis() - lastUltrasonicReading > ULTRASONIC_INTERVAL) {
    distance = readUltrasonicDistance();

    if (distance > 0 && distance < PROXIMITY_THRESHOLD) {
      if (debugMode) {
        debugPrint("⚠□ Proximity warning - object at " + String(distance, 1) + " cm");
      }
      digitalWrite(LED_YELLOW_PIN, HIGH);

      if (!buzzerActive) {
        digitalWrite(BUZZER_PIN, HIGH);
        delay(100);
        digitalWrite(BUZZER_PIN, LOW);
      }
    } else {
      digitalWrite(LED_YELLOW_PIN, LOW);
    }
```

```
    lastUltrasonicReading = millis();
  }

  // Send periodic status updates
  if (millis() - lastStatusUpdate > STATUS_UPDATE_INTERVAL) {
    sendStatusToFirestore("monitoring", "System running normally - v2.0", false);
    lastStatusUpdate = millis();
  }

  // MAIN FALL DETECTION LOGIC
  sensors_event_t a, g, temp;
  mpu.getEvent(&a, &g, &temp);

  float accel_magnitude = sqrt(a.acceleration.x * a.acceleration.x +
                               a.acceleration.y * a.acceleration.y +
                               a.acceleration.z * a.acceleration.z);

  float tiltAngle = calculateTiltAngle(a.acceleration.x, a.acceleration.y, a.acceleration.z);

  float gyro_magnitude = sqrt(g.gyro.x * g.gyro.x +
                              g.gyro.y * g.gyro.y +
                              g.gyro.z * g.gyro.z);

  // Detect movement for inactivity monitoring
  float accelChange = abs(accel_magnitude - lastAccelMagnitude);
  if (accelChange > MOVEMENT_THRESHOLD) {
    lastMovementTime = millis();
  }
  lastAccelMagnitude = accel_magnitude;

 // IMPROVED STATE MACHINE
 switch (currentState) {
   case MONITORING:
     digitalWrite(LED_GREEN_PIN, HIGH);
     digitalWrite(LED_RED_PIN, LOW);
     consecutiveTiltReadings = 0;

     if (tiltAngle > TILT_THRESHOLD) {
       consecutiveTiltReadings++;
       if (consecutiveTiltReadings >= REQUIRED_CONSECUTIVE_TILTS) {
         debugPrint("⚠ Significant tilt detected: " + String(tiltAngle, 2) + "° - monitoring for impact...");
         digitalWrite(LED_GREEN_PIN, LOW);
         currentState = TILT_DETECTED;
         fallDetectedTime = millis();
         consecutiveTiltReadings = 0;
       }
     } else {
       consecutiveTiltReadings = 0;
     }
     break;
```

```
    case IMPACT_DETECTED:
      debugPrint("🔍 Impact detected - monitoring for inactivity...");
      currentState = INACTIVITY_MONITORING;
      break;

    case INACTIVITY_MONITORING:
      digitalWrite(LED_RED_PIN, HIGH);
      digitalWrite(LED_YELLOW_PIN, LOW);

      if (millis() - fallDetectedTime > INACTIVITY_DURATION) {
        mpu.getEvent(&a, &g, &temp);
        float currentTiltAngle = calculateTiltAngle(a.acceleration.x, a.acceleration.y, a.acceleration.z);
        bool isInactive = (millis() - lastMovementTime > INACTIVITY_DURATION);


        debugPrint("📊 Inactivity analysis:");
        debugPrint("   Current tilt: " + String(currentTiltAngle, 2) + "° (threshold: " + String(ABNORMAL_ANGLE_THRESHOLD) + "°)");
        debugPrint("   Time since movement: " + String(millis() - lastMovementTime) + "ms");
        debugPrint("   Is inactive: " + String(isInactive ? "YES" : "NO"));

        if (currentTiltAngle > ABNORMAL_ANGLE_THRESHOLD && isInactive) {
          debugPrint("🚨 WHEELCHAIR FALL CONFIRMED! 🚨");
          debugPrint("🚨 Abnormal position + No movement detected! 🚨");
          debugPrint("🚨 ACTIVATING EMERGENCY PROTOCOL! 🚨");

          sendFallDataToFirestore(currentTiltAngle, a.acceleration.x, a.acceleration.y, a.acceleration.z, accel_magnitude);
          sendStatusToFirestore("EMERGENCY", "FALL DETECTED - Abnormal position with no movement", true);
          activateLocalAlarm();

        } else {
          debugPrint("✅ Position normalized or movement detected - false alarm");
          sendStatusToFirestore("false_alarm", "Potential fall detected but situation resolved", false);
        }

        currentState = MONITORING;
        digitalWrite(LED_RED_PIN, LOW);
      }
      break;
  }

  delay(100);
}

void updateBuzzer() {
  if (!buzzerActive) {
    return;
  }

  digitalWrite(LED_RED_PIN, HIGH);

  unsigned long currentTime = millis();


  if (currentTime - buzzerStartTime > BUZZER_DURATION) {
    digitalWrite(BUZZER_PIN, LOW);
    buzzerActive = false;
    debugPrint("🔔 Buzzer alarm complete");
    return;
  }

  if ((currentTime - buzzerStartTime) % (BUZZER_INTERVAL * 2) < BUZZER_INTERVAL) {
    digitalWrite(BUZZER_PIN, HIGH);
  } else {
    digitalWrite(BUZZER_PIN, LOW);
  }
}

// IMPROVED: Send fall data to notifications collection with better error handling
void sendFallDataToFirestore(float tiltAngle, float accelX, float accelY, float accelZ, float accelMagnitude) {
  if (WiFi.status() != WL_CONNECTED) {
    debugPrint("❌ Can't send fall data - no WiFi connection");
    return;
  }
```

```
  debugPrint("📤 Sending fall data to Firestore notifications...");

  HTTPClient http;
  String url = FIRESTORE_BASE_URL + "/notifications";
  http.begin(url);
  http.addHeader("Content-Type", "application/json");

  // Create Firestore document structure
  DynamicJsonDocument doc(1024);
  doc["fields"]["device_id"]["stringValue"] = deviceID;
  doc["fields"]["fallData"]["mapValue"]["fields"]["tiltAngle"]["doubleValue"] = tiltAngle;
  doc["fields"]["fallData"]["mapValue"]["fields"]["acceleration"]["mapValue"]["fields"]["x"]["doubleValue"] = accelX;
  doc["fields"]["fallData"]["mapValue"]["fields"]["acceleration"]["mapValue"]["fields"]["y"]["doubleValue"] = accelY;
  doc["fields"]["fallData"]["mapValue"]["fields"]["acceleration"]["mapValue"]["fields"]["z"]["doubleValue"] = accelZ;
  doc["fields"]["fallData"]["mapValue"]["fields"]["accelMagnitude"]["doubleValue"] = accelMagnitude;
  doc["fields"]["fallData"]["mapValue"]["fields"]["status"]["stringValue"] = "DEVICE_FALL";

  // Add ultrasonic proximity data
  doc["fields"]["obstacles"]["mapValue"]["fields"]["distance"]["doubleValue"] = distance;
  doc["fields"]["obstacles"]["mapValue"]["fields"]["unit"]["stringValue"] = "cm";

  // Add GPS location if available
  if (validGPS) {
    doc["fields"]["location"]["mapValue"]["fields"]["lat"]["doubleValue"] = gpsLat;
    doc["fields"]["location"]["mapValue"]["fields"]["long"]["doubleValue"] = gpsLng;
  } else {
    doc["fields"]["location"]["mapValue"]["fields"]["lat"]["doubleValue"] = -6.175381;
    doc["fields"]["location"]["mapValue"]["fields"]["long"]["doubleValue"] = 106.9508;
  }

  // Add timestamp
  doc["fields"]["createdAt"]["timestampValue"] = getCurrentTimestamp();
  doc["fields"]["updatedAt"]["timestampValue"] = getCurrentTimestamp();
  doc["fields"]["status"]["stringValue"] = "sent";


  String json;
  serializeJson(doc, json);

  debugPrint("📡 Sending to: " + url);
  if (debugMode) {
    debugPrint("📄 Payload size: " + String(json.length()) + " bytes");
  }

  int httpCode = http.POST(json);

  if(httpCode == HTTP_CODE_OK || httpCode == 200) {
    debugPrint("✅ Fall data sent to Firestore notifications successfully!");
    String response = http.getString();
    if (debugMode) {
      debugPrint("📥 Response: " + response.substring(0, 100) + "...");
    }
  } else {
    debugPrint("❌ Failed to send fall data. HTTP Code: " + String(httpCode));
    String errorResponse = http.getString();
    debugPrint("❌ Error response: " + errorResponse.substring(0, 200));
  }

  http.end();
}

// IMPROVED: Update device status with better error handling
void sendStatusToFirestore(String status, String message, bool isEmergency) {
  if (WiFi.status() != WL_CONNECTED) {
    debugPrint("❌ Can't send status - no WiFi connection");
    return;
  }

  debugPrint("📤 Updating device status: " + status);

  HTTPClient http;
```

```
// Try to update existing document first
String deviceDocId = String(deviceID) + "_status";
String updateUrl = FIRESTORE_BASE_URL + "/devices/" + deviceDocId;

http.begin(updateUrl + "?updateMask.fieldPaths=status&updateMask.fieldPaths=message&updateMask.
http.addHeader("Content-Type", "application/json");

DynamicJsonDocument doc(1024);
doc["fields"]["device_id"]["stringValue"] = deviceID;
doc["fields"]["status"]["stringValue"] = status;
doc["fields"]["message"]["stringValue"] = message;
doc["fields"]["isEmergency"]["booleanValue"] = isEmergency;

// Add fall data if emergency
if (isEmergency && status == "EMERGENCY") {
  sensors_event_t a, g, temp;
  mpu.getEvent(&a, &g, &temp);
  float tiltAngle = calculateTiltAngle(a.acceleration.x, a.acceleration.y, a.acceleration.z);

  doc["fields"]["fallData"]["mapValue"]["fields"]["tiltAngle"]["doubleValue"] = tiltAngle;
  doc["fields"]["fallData"]["mapValue"]["fields"]["acceleration"]["mapValue"]["fields"]["x"]["doubleValue"] = a.acceleration.x;
  doc["fields"]["fallData"]["mapValue"]["fields"]["acceleration"]["mapValue"]["fields"]["y"]["doubleValue"] = a.acceleration.y;
  doc["fields"]["fallData"]["mapValue"]["fields"]["acceleration"]["mapValue"]["fields"]["z"]["doubleValue"] = a.acceleration.z;
  doc["fields"]["fallData"]["mapValue"]["fields"]["status"]["stringValue"] = "DEVICE_FALL";
}

// Add ultrasonic proximity data
doc["fields"]["sensors"]["mapValue"]["fields"]["ultrasonic"]["mapValue"]["fields"]["distance"]["doubleValue"] = distance;
doc["fields"]["sensors"]["mapValue"]["fields"]["ultrasonic"]["mapValue"]["fields"]["unit"]["stringValue"] = "cm";

// Add GPS location if available
if (validGPS) {
  doc["fields"]["location"]["mapValue"]["fields"]["lat"]["doubleValue"] = gpsLat;
  doc["fields"]["location"]["mapValue"]["fields"]["long"]["doubleValue"] = gpsLng;
} else {
  doc["fields"]["location"]["mapValue"]["fields"]["lat"]["doubleValue"] = -6.175381;
  doc["fields"]["location"]["mapValue"]["fields"]["long"]["doubleValue"] = 106.9508;
}


// Add timestamp
doc["fields"]["updatedAt"]["timestampValue"] = getCurrentTimestamp();
doc["fields"]["lastPing"]["timestampValue"] = getCurrentTimestamp();

String json;
serializeJson(doc, json);

int httpCode = http.PATCH(json);

if(httpCode == HTTP_CODE_OK || httpCode == 200) {
  debugPrint("✅ Device status updated successfully");
} else {
  debugPrint("⚠️ Update failed, trying to create new document. Error: " + String(httpCode));


  // Try to create new document
  http.end();
  String createUrl = FIRESTORE_BASE_URL + "/devices";
  http.begin(createUrl);
  http.addHeader("Content-Type", "application/json");

  // Add creation timestamp
  doc["fields"]["createdAt"]["timestampValue"] = getCurrentTimestamp();

  String createJson;
  serializeJson(doc, createJson);
```

```
    int createCode = http.POST(createJson);
    if(createCode == HTTP_CODE_OK || createCode == 200) {
      debugPrint("✅ Device status created successfully");
    } else {
      debugPrint("❌ Failed to create device status. Code: " + String(createCode));
      String errorResponse = http.getString();
      debugPrint("❌ Error: " + errorResponse.substring(0, 200));
    }
  }

  http.end();
}

String getCurrentTimestamp() {
  // Get current time in ISO format
  time_t now = time(0);
  struct tm* timeinfo = gmtime(&now);

  char timestamp[30];
  strftime(timestamp, sizeof(timestamp), "%Y-%m-%dT%H:%M:%SZ", timeinfo);

  return String(timestamp);
}

void activateLocalAlarm() {
  buzzerActive = true;
  buzzerStartTime = millis();
  debugPrint("🚨 LOCAL ALARM ACTIVATED - BUZZER SOUNDING!");

  // Flash all LEDs for emergency
  for (int i = 0; i < 5; i++) {
    digitalWrite(LED_RED_PIN, HIGH);
    digitalWrite(LED_YELLOW_PIN, HIGH);
    digitalWrite(LED_GREEN_PIN, HIGH);
    delay(200);
    digitalWrite(LED_RED_PIN, LOW);
    digitalWrite(LED_YELLOW_PIN, LOW);
    digitalWrite(LED_GREEN_PIN, LOW);
    delay(200);
  }
}

void testLEDs() {
  debugPrint("💡 Testing LED indicators...");

  debugPrint("   - Green LED (normal status)");
  digitalWrite(LED_GREEN_PIN, HIGH);
  delay(500);
  digitalWrite(LED_GREEN_PIN, LOW);

  debugPrint("   - Yellow LED (warning)");
  digitalWrite(LED_YELLOW_PIN, HIGH);
  delay(500);
  digitalWrite(LED_YELLOW_PIN, LOW);

  debugPrint("   - Red LED (emergency)");
  digitalWrite(LED_RED_PIN, HIGH);
  delay(500);
  digitalWrite(LED_RED_PIN, LOW);
```

```
  debugPrint("✓ LED test complete");
}

void setupGPS() {
  debugPrint("🛰️ Initializing GPS module...");
  GPSSerial.begin(GPS_BAUD, SERIAL_8N1, GPS_RX_PIN, GPS_TX_PIN);
  delay(1000);
  debugPrint("✓ GPS module initialized on Serial1");
  debugPrint("⏳ Waiting for GPS signal (can take several minutes outdoors)...");
}

void printGPSDebugInfo() {
  if (!debugMode) return;

  debugPrint("=== GPS STATUS ===");
  debugPrint("Characters processed: " + String(gps.charsProcessed()));
  debugPrint("Sentences with fix: " + String(gps.sentencesWithFix()));
  debugPrint("Failed checksum: " + String(gps.failedChecksum()));
  debugPrint("Passed checksum: " + String(gps.passedChecksum()));

  if (gps.location.isValid()) {
    debugPrint("Location: " + String(gps.location.lat(), 6) + ", " + String(gps.location.lng(), 6));
  } else {
    debugPrint("Location: INVALID");
  }

  if (gps.satellites.isValid()) {
    debugPrint("Satellites: " + String(gps.satellites.value()));
  } else {
    debugPrint("Satellites: INVALID");
  }
  debugPrint("==================");
}
```