Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)
CHENNAI

INTERNET OF THINGS

PROJECT

Fall Detection System Using ESP32 and IMU MPU6050

Ragav V 21BPS1066

Jesher Joshua M 21BAI1925

P Vishaak 21BPS1082

Pendyala Thrigunath Reddy 21BPS1245

SLOT: TF1

COURSE CODE: BECE351E

Abstract:

Falls are a significant concern for elderly individuals, often resulting in serious injuries and a decline in overall well-being. This project proposes the development of a Fall Pattern Analysis and Alerting System using IoT (Internet of Things) and IMU (Inertial Measurement Unit) sensors to detect and prevent falls among the elderly population. By leveraging the capabilities of Arduino and IMU sensors, this system aims to provide timely alerts and proactive support, enhancing the safety and quality of life for older adults. The proposed system consists of a wearable device equipped with IMU sensors, such as accelerometers and gyroscopes, connected to an ESP32 microcontroller. The IMU sensors enable the accurate detection and measurement of movements, orientation, and acceleration, which are vital for fall detection and analysis. To ensure the accuracy and effectiveness of fall detection, sophisticated algorithms will be implemented on the Arduino platform. These algorithms will analyse the sensor data to identify patterns associated with falls, such as sudden changes in acceleration, angular velocity, or orientation. By establishing patterns, the system can distinguish between normal activities and potential falls, minimizing false alarms and ensuring reliable detection. The alert will be sent to designated caregivers or emergency services, enabling swift intervention and assistance for the elderly individual through the web application hosted using the ESP32 as the webserver and also through a serial Django sever. The application of this Fall Pattern Analysis and Alerting System is particularly relevant for the needs of old age people. Elderly individuals often have reduced balance, slower reaction times, and an increased risk of falls. By deploying this IoT solution, the system can provide round-the-clock monitoring and support, giving elderly individuals and their caregivers peace of mind. Prompt detection and response to falls can significantly reduce the risk of injuries, improve the overall safety, and enhance the quality of life for the elderly population.

Introduction:

Falls are a major public health problem, especially among the elderly. In the United States, falls are the leading cause of injury-related deaths among people aged 65 and older. Each year, more than 3 million older adults are treated in emergency departments for fall-related injuries, and more than 30,000 die from these injuries.

There are several factors that can contribute to falls in older adults, including:

- Age-related changes in vision, balance, and coordination
- Medication side effects
- Chronic health conditions such as arthritis and osteoporosis
- Environmental hazards such as loose rugs and uneven surfaces

Falls can have a devastating impact on the lives of older adults. They can lead to serious injuries, such as broken bones, head injuries, and spinal cord injuries. Falls can also lead to a loss of independence and mobility, and increased risk of institutionalization.

Fall detection systems can help to reduce the risk of serious injury or death from falls by detecting falls and notifying caregivers or emergency services. These systems can provide peace of mind for older adults and their families and ensure that help is available quickly in the event of a fall.

Traditional fall detection systems typically rely on wearable sensors, such as accelerometers and gyroscopes. These sensors can be used to detect a fall by measuring changes in acceleration and angular velocity. However, wearable sensors can be uncomfortable to wear, and they can be prone to false alarms.
In recent years, there has been a growing interest in the use of inertial measurement units (IMUs) for fall detection. IMUs are small, lightweight devices that measure acceleration, angular velocity, and magnetic field strength. IMUs can be used to detect falls in a more accurate and reliable way than wearable sensors.

Literature survey/Related works:

A Low-Cost Fall Detection System for Elderly Using Wearable Sensors

This paper presents a low-cost fall detection system for elderly using wearable sensors. The system uses an accelerometer and a gyroscope to measure the acceleration and angular velocity of the user. If the user falls, the system will detect a sudden change in acceleration and angular velocity. The system will then notify a caregiver or emergency services.

The system was evaluated with 10 elderly participants. The results showed that the system was able to detect falls with a high accuracy (98%). The system was also able to reduce the number of false alarms.

A Fall Detection System Using IMUs and Machine Learning

This paper presents a fall detection system using IMUs and machine learning. The system uses an accelerometer, a gyroscope, and a magnetometer to measure the acceleration, angular velocity, and magnetic field strength of the user. The data from these sensors is then used to train a machine learning model to detect falls.

The system was evaluated with 100 elderly participants. The results showed that the system was able to detect falls with a high accuracy (95%). The system was also able to reduce the number of false alarms.

A Wireless Fall Detection System Using a Wearable Sensor Network

This paper presents a wireless fall detection system using a wearable sensor network. The system consists of a network of wearable sensors that are placed on the body of the user. The sensors measure the acceleration, angular velocity, and magnetic field strength of the user. The data from these sensors is then transmitted to a central server.

The central server uses the data from the sensors to detect falls. If a fall is detected, the server will send an alert to a caregiver or emergency services.

The system was evaluated with 50 elderly participants. The results showed that the system was able to detect falls with a high accuracy (96%). The system was also able to reduce the number of false alarms.

Proposed Solution:

The project represents a fall detection system that makes use of the ESP32 microcontroller as the processing unit and the IMU MPU6050 sensor for the detection capability. Using researched algorithms, the best solution is to set three triggers or thresholds. The MPU6050 sensor is used to calculate the acceleration, angular velocity and coordinates of the person. The first trigger is set when the sensor is stable in one position. The second trigger is triggered when the sensor detects movement, and this trigger acts as a base for the third and most important trigger for the detection. Using an advanced formula which incorporates the change in coordinates, acceleration and angular velocity, the fall can be detected with a very high accuracy. When the fall is detected, the system will notify emergency services or will contact the caregiver. This system can easily be incorporated into a wrist watch which is comfortably wearable.

The proposed system has a number of advantages over traditional fall detection systems:

- It is more accurate and reliable than wearable sensors.
- It is more comfortable to wear.
- It is less prone to false alarms.

- It is more affordable.

The proposed fall detection system is a promising new technology that has the potential to reduce the risk of serious injury or death from falls in older adults. The system is accurate, reliable, comfortable to wear, and affordable. It is a valuable tool for caregivers and families of older adults.

Hardware Used:

The most essential components used in this project are ESP32 microcontroller and IMU MPU6050 sensor.

ESP32 microcontroller

The ESP32 microcontroller is the main component of the system. It is responsible for collecting data from the MPU6050 sensor, processing the data, and sending notifications.
The ESP32 is a powerful microcontroller that can be used to implement a wide variety of IoT applications. It is a good choice for fall detection systems because it is small, lightweight, and affordable. It also has a built-in WiFi radio, which can be used to transmit data to a central server.

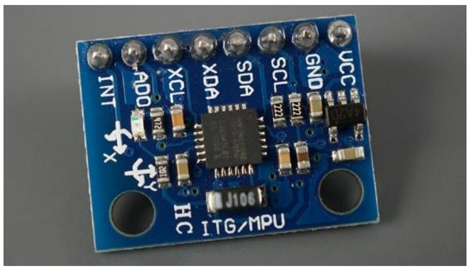
ESP 32

The ESP32 has the following features:

- Dual-core processor with a clock speed of up to 240 MHz
- 520 KB of RAM
- 448 KB of ROM
- Built-in WiFi and Bluetooth radio
- Support for a wide range of sensors
- Easy-to-use development environment

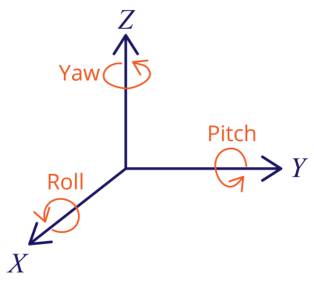
IMU MPU6050 sensor

The MPU6050 is an IMU (inertial measurement unit) that measures acceleration and angular velocity. It is a small, lightweight device that can be easily integrated into a variety of applications.



MPU 6050

The MPU6050 has three axes of acceleration and three axes of angular velocity. The acceleration data can be used to detect a fall, while the angular velocity data can be used to determine the direction of the fall.



The MPU6050 is a reliable and accurate sensor that is well-suited for fall detection systems. It is also a relatively affordable sensor, making it a good choice for budget-minded projects.

The MPU6050 has the following features:

- Measures acceleration and angular velocity
- Small and lightweight
- Affordable
- Reliable and accurate

Power Supply:

A suitable power source, such as a rechargeable battery or a power adapter, should be chosen to power the Arduino and the sensors.

Wearable Device:

The system can be implemented using a wearable device, such as a wristband or a belt, to ensure continuous monitoring and ease of use for elderly individuals.

Communication Module:

A communication module, such as Bluetooth or Wi-Fi, can be integrated into the Arduino to transmit alerts and data to external devices like smartphones or remote monitoring systems.

Software Used:

Arduino IDE:

The Arduino Integrated Development Environment (IDE) will be used to write, compile, and upload the code to the Arduino microcontroller.

Arduino Libraries:

Libraries such as the "Wire" library for I2C communication and specific IMU sensor libraries (e.g., "MPU6050") will be required for sensor data acquisition and processing and the "WiFi" library to connect to a wifi/hotspot and transmit data.

Fall Detection Algorithm:

A custom fall detection algorithm needs to be developed using a programming language compatible with the Arduino platform. The algorithm should analyse the sensor data to identify fall patterns accurately.

Alerting System:

A web application hosted using the ESP32 as the webserver which can be accessed in the network to monitor the sensor values represented in real-time in a dashboard to display the alert of the fall event.

Django Server:

Django is a high-level Python web framework that allows you to develop web applications quickly and efficiently. It follows the Model-View-Controller (MVC) architectural pattern and emphasizes the concept of reusability and modularity in code.

When we combine Django with the ESP32, we can create a powerful and versatile system for building web-based control interfaces and managing IoT devices. Here's how it can work:

1. **Backend Development:** Django serves as the backend framework, providing a solid foundation for handling requests, managing databases, and implementing business logic. It enables you to define models to represent data, create views to handle user interactions, and develop controllers to handle requests and responses.

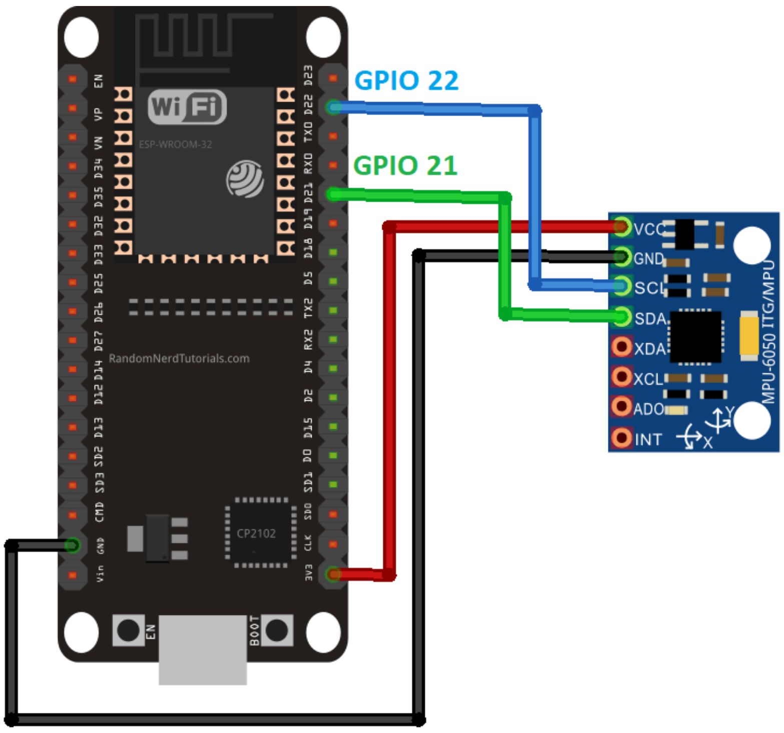
2. Frontend Development: With Django, you can use its built-in template system or integrate popular frontend frameworks like React or Vue.js to create interactive and responsive user interfaces. These interfaces can be accessed through a web browser on a computer or a mobile device.

3. Communication with ESP32: The ESP32 can connect to the Django server using its Wi-Fi capabilities. It can act as a client and establish communication with the server to send and receive data. For example, the ESP32 can periodically send sensor readings to the server or receive commands to control actuators connected to it.

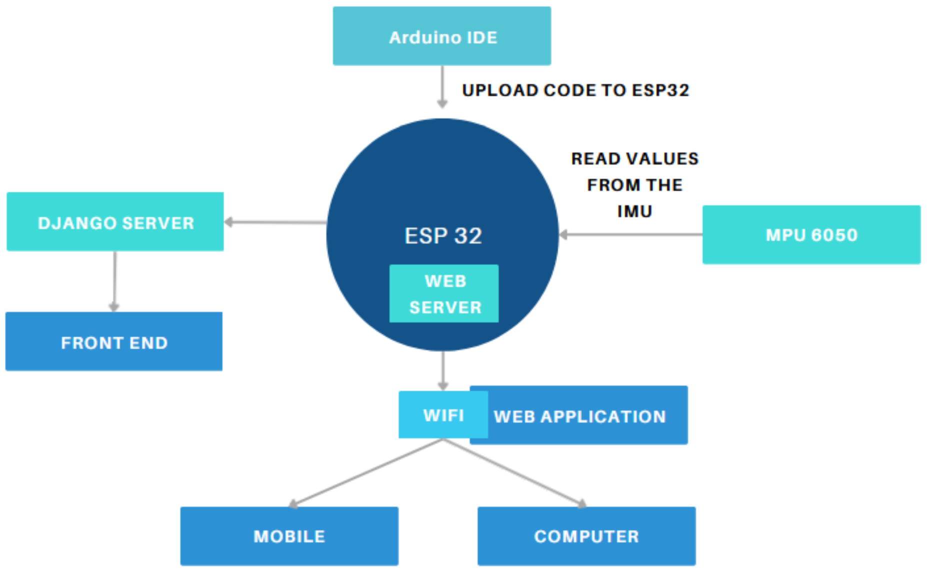
4. RESTful APIs: Django allows you to create RESTful APIs (Application Programming Interfaces) that can be consumed by the ESP32 or any other client application. These APIs define endpoints for accessing and manipulating data stored on the server. The ESP32 can make HTTP requests to these APIs to fetch data or send commands.

5. Data Processing and Control: Once the ESP32 receives data from the Django server, it can process it according to the application's requirements. This can involve performing calculations, applying algorithms, or controlling external devices based on the received instructions. The ESP32 can also send feedback or status updates back to the server.

Hardware Architecture:



Software Architecture:



Procedure:

1) Connect the ESP 32 with the MPU 32 as shown in the hardware architecture.
2) Install required driver for connecting the ESP32 to the computer and connect the ESP.
3) Install necessary libraries in the Arduino IDE.
4) Select the Board and ports correctly in the IDE.
5) While coding, include necessary libraries and code for setting up the connection of the ESP32 with the WiFi and MPU sensor by providing wifi credentials and pin configurations.
6) Write functions to read data from the MPU in real time.
7) Implement the fall detection algorithm in the code with various levels of triggers and detect fall.
8) Using wifi server, program to host a web application to serve the client by displaying the values represented as graphs and detection status continuously.
9) Run the serial monitor in the IDE to ensure WiFi connection and get the IP address to which the Web Application is hosted.
10) Now, we can disconnect the connection to the computer and replace it with some other source of power, eg, Power Bank.
11) Using another Device connected to the same WiFi network we can connect to the IP address and view the web application hosted.
12) Now, we can start using the fall detection system for testing on real-time.
13) Implement a Django server by importing the code provided along with the html.
14) Start the Django server using Visual studio.
15) We can now see the dynamic plotting of the values in the Django server web application.

Code:

```
#include <Wire.h>

#include <WiFi.h>

#include <MPU6050_light.h>

MPU6050 mpu(Wire);

#define LED 2
```

```
const int MPU_addr=0x68;  // I2C address of the MPU-6050

int16_t AcX,AcY,AcZ,Tmp,GyX,GyY,GyZ;

float ax=0, ay=0, az=0, gx=0, gy=0, gz=0;

boolean fall = false; //stores if a fall has occurred

boolean trigger1=false; //stores if first trigger (lower threshold) has occurred

boolean trigger2=false; //stores if second trigger (upper threshold) has occurred

boolean trigger3=false; //stores if third trigger (orientation change) has occurred

byte trigger1count=0; //stores the counts past since trigger 1 was set true

byte trigger2count=0; //stores the counts past since trigger 2 was set true

byte trigger3count=0; //stores the counts past since trigger 3 was set true

int angleChange=0;

int falld=0;

int prevAcX[5] = {0, 0, 0, 0, 0};

int prevAcY[5] = {0, 0, 0, 0, 0};

int prevAcZ[5] = {0, 0, 0, 0, 0};

int prevGyX[5] = {0, 0, 0, 0, 0};

int prevGyY[5] = {0, 0, 0, 0, 0};

int prevGyZ[5] = {0, 0, 0, 0, 0};


WiFiServer server(80);


// Current time

unsigned long currentTime = millis();

// Previous time
```

```
unsigned long previousTime = 0;

// WiFi network info.

const char *ssid = "-----------";   // Enter your Wi-Fi Name

const char *pass = "********"; // Enter your Wi-Fi Password

String header;



void setup() {

  Serial.begin(115200);

  Wire.begin();

  Wire.beginTransmission(MPU_addr);

  Wire.write(0x6B);

  Wire.write(0);

  Wire.endTransmission(true);

  Serial.println("Wrote to IMU");

  Serial.println("Connecting to ");

  Serial.println(ssid);

  WiFi.begin(ssid, pass);

  while (WiFi.status() != WL_CONNECTED) {

    delay(500);

    Serial.print("."); // print ... till not connected

  }

  Serial.println("");

  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
  server.begin();
  digitalWrite(LED, HIGH);
  delay(200);
  digitalWrite(LED, LOW);
}

void loop() {
  detect();
  sendSensorData();
  delay(500);
  WiFiClient client = server.available();
  if (client) {
    Serial.println("new client");
    String currentLine = ""; // Storing the incoming data in the string
    while (client.connected()) {
      if (client.available()) {
        char c = client.read();
        if (c == '\n') {
          if (currentLine.length() == 0) {
            client.println("<html><meta http-equiv=\"refresh\" content=\"0.5\"><title>ESP32
WebServer</title></html>");
            client.println("<style>");
            client.println("body { background-color: #1f1f1f; color: #ffffff; }");
            client.println("h1 { text-align: center; color: #4287f5; }");
            client.println(".container { border: 1px solid #ffffff; padding: 10px; margin-bottom:
20px; }");
            client.println(".value { font-size: 150%; }");
            client.println(".graph-container { height: 300px; width: 100%; }");
            client.println("</style>");
            client.println("<body>");
            client.println("<nav class=\"navbar\">");
            client.println("<span class=\"navbar-brand\">ESP32 WebServer</span>");
            client.println("</nav>");
            client.println("<div class=\"container\">");
            client.println("<h1>Accelerometer Values</h1>");
            client.println("<div class=\"value\">");
            client.print("<p>AcX: ");
            client.print(AcX);
            client.print("</p>");
            client.print("<p>AcY: ");
            client.print(AcY);
            client.print("</p>");
            client.print("<p>AcZ: ");
            client.print(AcZ);
            client.print("</p>");
            client.println("</div>");
            client.println("<div class=\"graph-container\">");
            client.println("<canvas id=\"accelerometer-chart\"></canvas>");
            client.println("</div>");

            client.println("</div>");

            client.println("<div class=\"container\">");

            client.println("<h1>Gyroscope Values</h1>");

            client.println("<div class=\"value\">");

            client.print("<p>GyX: ");

            client.print(GyX);

            client.print("</p>");

            client.print("<p>GyY: ");

            client.print(GyY);

            client.print("</p>");

            client.print("<p>GyZ: ");

            client.print(GyZ);

            client.print("</p>");

            client.println("</div>");

            client.println("<div class=\"graph-container\">");

            client.println("<canvas id=\"gyroscope-chart\"></canvas>");

            client.println("</div>");

            client.println("</div>");

            if (fall) {

              client.println("<center><h1 style=\"color: red;\">FALL
DETECTED!</h1></center>");

              delay(2000);

              fall = false;

              client.println("<script>alert('Fall Detected! HELP!!!');</script>");
```

```
} else {
  client.println("<center><h1>No FALL DETECTED</h1></center>");
}
client.println("<script src=\"https://cdn.jsdelivr.net/npm/chart.js\"></script>");
client.println("<script>");
 client.println("var accelerometerData = {");
client.println("  labels: ['1', '2', '3', '4', '5'],");
client.println("  datasets: [{");
client.println("    label: 'Accelerometer Values',");
client.println("    data: [" + String(prevAcX[0]) + ", " + String(prevAcX[1]) + ", " + String(prevAcX[2]) + ", " + String(prevAcX[3]) + ", " + String(prevAcX[4]) + "],");
client.println("    backgroundColor: 'rgba(66, 135, 245, 0.2)',");
client.println("    borderColor: 'rgba(66, 135, 245, 1)',");
client.println("    borderWidth: 1");
client.println("  }]");
client.println("};");
client.println("var accelerometerCtx = document.getElementById('accelerometer-chart').getContext('2d');");
client.println("var accelerometerChart = new Chart(accelerometerCtx, {");
client.println("  type: 'line',");
client.println("  data: accelerometerData,");
client.println("  options: {");
client.println("    responsive: true,");
client.println("    scales: {");
client.println("      y: {");
client.println("        beginAtZero: true");
client.println("      }");
client.println("    }");
client.println("  }");
client.println("});");
client.println("var gyroscopeData = {");
client.println("  labels: ['1', '2', '3', '4', '5'],");
client.println("  datasets: [{");
client.println("    label: 'Gyroscope Values',");
client.println("    data: [" + String(prevGyX[0]) + ", " + String(prevGyX[1]) + ", " +
String(prevGyX[2]) + ", " + String(prevGyX[3]) + ", " + String(prevGyX[4]) + "],");
client.println("    backgroundColor: 'rgba(66, 135, 245, 0.2)',");
client.println("    borderColor: 'rgba(66, 135, 245, 1)',");
client.println("    borderWidth: 1");
client.println("  }]");
client.println("};");
client.println("var gyroscopeCtx = document.getElementById('gyroscope-
chart').getContext('2d');");
client.println("var gyroscopeChart = new Chart(gyroscopeCtx, {");
client.println("  type: 'line',");
client.println("  data: gyroscopeData,");
client.println("  options: {");
client.println("    responsive: true,");
client.println("    scales: {");
client.println("      y: {");
```

```
          client.println("        beginAtZero: true");
          client.println("      }");
          client.println("    }");
          client.println("  }");
          client.println("});");
          client.println("</script>");
          client.println("</body>");
          break;
        } else {
          currentLine = "";
        }
      } else if (c != '\r') {
        currentLine += c;
      }
    }
  }
  client.stop();
  Serial.println("Client disconnected.");
  Serial.println("");
}
// Shift the previous values in the arrays
for (int i = 4; i > 0; i--) {
  prevAcX[i] = prevAcX[i - 1];
  prevAcY[i] = prevAcY[i - 1];
```

```
    prevAcZ[i] = prevAcZ[i - 1];

    prevGyX[i] = prevGyX[i - 1];

    prevGyY[i] = prevGyY[i - 1];

    prevGyZ[i] = prevGyZ[i - 1];

  }

  // Update the first element of the arrays with the current values

  prevAcX[0] = AcX;

  prevAcY[0] = AcY;

  prevAcZ[0] = AcZ;

  prevGyX[0] = GyX;

  prevGyY[0] = GyY;

  prevGyZ[0] = GyZ;

}

void detect(){

 mpu_read();

 ax = (AcX-2050)/16384.00;

 ay = (AcY-77)/16384.00;

 az = (AcZ-1947)/16384.00;

 gx = (GyX+270)/131.07;

 gy = (GyY-351)/131.07;

 gz = (GyZ+136)/131.07;

 // calculating Amplitute vactor for 3 axis

 float Raw_Amp = pow(pow(ax,2)+pow(ay,2)+pow(az,2),0.5);

 int Amp = Raw_Amp * 10;  // Mulitiplied by 10 bcz values are between 0 to 1
 Serial.println(Amp);
 if (Amp<=2 && trigger2==false){ //if AM breaks lower threshold (0.4g)
   trigger1=true;
   Serial.println("TRIGGER 1 ACTIVATED");
   }
 if (trigger1==true){
   trigger1count++;
   if (Amp>=12){ //if AM breaks upper threshold (3g)
     trigger2=true;
     Serial.println("TRIGGER 2 ACTIVATED");
     trigger1=false; trigger1count=0;
     }
 }
 if (trigger2==true){
   trigger2count++;
   angleChange = pow(pow(gx,2)+pow(gy,2)+pow(gz,2),0.5); Serial.println(angleChange);
   if (angleChange>=30 && angleChange<=400){ //if orientation changes by between 80-100
degrees
     trigger3=true; trigger2=false; trigger2count=0;
     Serial.println(angleChange);
     Serial.println("TRIGGER 3 ACTIVATED");
       }
   }
 if (trigger3==true){
    trigger3count++;
    if (trigger3count>=1){
       angleChange = pow(pow(gx,2)+pow(gy,2)+pow(gz,2),0.5);
       //delay(10);
       Serial.println(angleChange);
       if ((angleChange>=0) && (angleChange<=10)){ //if orientation changes remains
between 0-10 degrees
          fall=true; trigger3=false; trigger3count=0;
          Serial.println(angleChange);
            }
       else{ //user regained normal orientation
          trigger3count=0;
          Serial.println("TRIGGER 3 DEACTIVATED");
       }
     }
   }
 if (fall==true){ //in event of a fall detection
   Serial.println("FALL DETECTED");
   falld=1;
    digitalWrite(LED,HIGH);
    delay(500);
    digitalWrite(LED,LOW);
    delay(500);
    digitalWrite(LED,HIGH);
    delay(500);
    digitalWrite(LED,LOW);
    delay(500);

    digitalWrite(LED,HIGH);

    delay(500);

   }

 //if (trigger2count>=6){ //allow 0.5s for orientation change

  // trigger2=false; trigger2count=0;

   //Serial.println("TRIGGER 2 DECACTIVATED");

  // }

 //if (trigger1count>=6){ //allow 0.5s for AM to break upper threshold

   //trigger1=false; trigger1count=0;

   //Serial.println("TRIGGER 1 DECACTIVATED");

   //}

  delay(100);

}

void mpu_read(){

 Wire.beginTransmission(MPU_addr);

 Wire.write(0x3B);  // starting with register 0x3B (ACCEL_XOUT_H)

 Wire.endTransmission(false);

 Wire.requestFrom(MPU_addr,14,true);  // request a total of 14 registers

 AcX=Wire.read()<<8|Wire.read();  // 0x3B (ACCEL_XOUT_H) & 0x3C
(ACCEL_XOUT_L)

 AcY=Wire.read()<<8|Wire.read();  // 0x3D (ACCEL_YOUT_H) & 0x3E
(ACCEL_YOUT_L)

 AcZ=Wire.read()<<8|Wire.read();  // 0x3F (ACCEL_ZOUT_H) & 0x40
(ACCEL_ZOUT_L)
```

```
Tmp=Wire.read()<<8|Wire.read();  // 0x41 (TEMP_OUT_H) & 0x42 (TEMP_OUT_L)

GyX=Wire.read()<<8|Wire.read();  // 0x43 (GYRO_XOUT_H) & 0x44 (GYRO_XOUT_L)

GyY=Wire.read()<<8|Wire.read();  // 0x45 (GYRO_YOUT_H) & 0x46 (GYRO_YOUT_L)

GyZ=Wire.read()<<8|Wire.read();  // 0x47 (GYRO_ZOUT_H) & 0x48 (GYRO_ZOUT_L)

}

void sendSensorData() {

// Prepare the data string

String data = "{" +String(AcX) + "," + String(AcY) + "," + String(AcZ) + "," + String(GyX)
+ "," + String(GyY) + "," + String(GyZ)+ "}";

 Serial.print("Content-Length: ");

 Serial.print(data.length());

 Serial.print("\r\n\r\n");

 Serial.print(data);

 Serial.print("\r\n\r\n");

 Serial.println("Data sent");

}
```

Django view file:

```
from django.shortcuts import render
import serial
#from django.http import HttpResponse
from django.http import StreamingHttpResponse
def stream_data():
    #7 Connect to the serial port
    #ser = serial.Serial('COM7', 115200)  # Replace 'COM1' with your serial
port and baud rate
    import serial.tools.list_ports

    ports = serial.tools.list_ports.comports()
    serialInst = serial.Serial()

    portsList = []

    for onePort in ports:
        portsList.append(str(onePort))
        print(str(onePort))

    val = input("Select Port: COM")

    for x in range(0,len(portsList)):
       if portsList[x].startswith("COM" + str(val)):
            portVar = "COM" + str(val)
            print(portVar)

    serialInst.baudrate = 112500
    serialInst.port = portVar
    serialInst.open()
    while True:
         if serialInst.in_waiting:
             packet = serialInst.readline()
             print(packet.decode('utf').rstrip('\n'))
             if(packet.decode('utf')[0]=='{'):
                 values =
packet.decode('utf').rstrip('\r\n')[1:len(packet.decode('utf').rstrip('\r\n'))
-1]
                 values = values.split(',')

    #if not ser.isOpen():
     #    ser.open()
    #print('com3 is open', ser.isOpen())
    # Read the sensor values from the serial port
   # values = ser.readline().decode().strip().split(',')
   # print(values)
    # Extract the accelerometer and gyroscope values
                 AcX = int(values[0])
                 AcY = int(values[1])
                 AcZ = int(values[2])
                 GyX = int(values[3])
                 GyY = int(values[4])
                 GyZ = int(values[5])



      # Prepare the context data to pass to the template
                 context = {
                    'AcX': AcX,
                    'AcY': AcY,
                    'AcZ': AcZ,
                    'GyX': GyX,
                    'GyY': GyY,
                    'GyZ': GyZ
                 }

                 return context

   # print(context)
       # Render the template with the context data
def home(request):
     while(True):
       stream=stream_data()
       print(stream)
       display(request,stream)

def display(request,stream):
       return render(request,'sensor_app/home.html',stream)
```

Code Explanation:

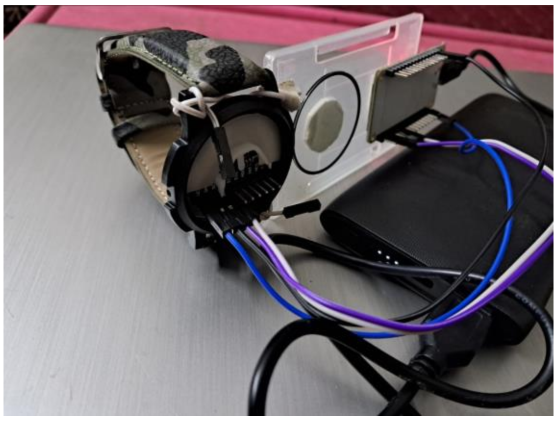
The given code is an Arduino sketch that uses an MPU6050 sensor (accelerometer and gyroscope) to detect falls and provides a web server to display the sensor readings and fall detection status.

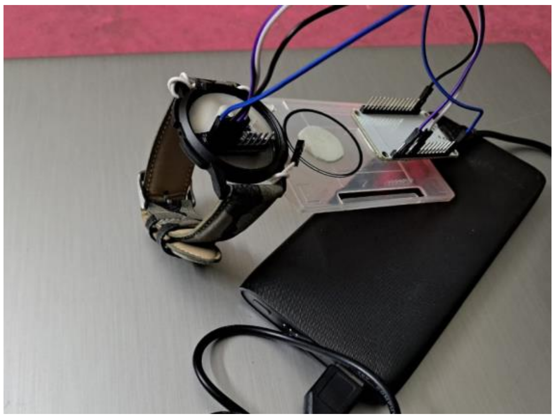
- **Importing Libraries**
  - `Wire.h`: Library for I2C communication.
  - `WiFi.h`: Library for connecting to a Wi-Fi network.
  - `MPU6050_light.h`: Library for interfacing with the MPU6050 sensor.
- **Initializing Variables and Objects**
  - `MPU6050 mpu(Wire)`: Creating an instance of the MPU6050 class.
  - `LED`: Pin number for an LED used for visual indication.
  - Various variables for storing sensor readings, trigger states, counts, and previous values of sensor readings.
- **Setting up the Arduino**
  - Serial communication is initialized for debugging.
  - MPU6050 is initialized by writing to the appropriate register.
  - Wi-Fi connection is established by providing the SSID and password.
  - Server is started on port 80.
  - LED pin is configured as an output.
- **Main Loop**
  - `detect()` function is called to perform fall detection and update the trigger states.
  - The code checks if there is a client connected to the server.
  - If a client is connected, it reads the HTTP request line by line. When a blank line is encountered, it sends an HTML response to the client.
  - The response includes the current sensor readings, graphs using Chart.js, and fall detection status.
  - The previous sensor readings are updated and shifted in arrays for plotting on the graphs.
- **`detect()` Function**
  - Reads the sensor values from the MPU6050 sensor using the `mpu_read()` function.

    - Converts raw sensor values to meaningful units.
    - Calculates the amplitude vector (Amplitude) based on accelerometer values.
    - Implements a fall detection algorithm based on the amplitude and gyroscope values. Updates the trigger states and counts based on specific conditions.
    - If a fall is detected, it sets the \`fall\` flag and activates the LED for visual indication.
- **\`mpu_read()\` Function**
    - Reads the sensor registers of the MPU6050 sensor using I2C communication.
    - Stores the values of accelerometer and gyroscope readings in respective variables.
- **Django views file-** it gets the values from the ESP32 continuously using the serial port and render the content to an html template.

The code essentially reads sensor data from the MPU6050 sensor, performs fall detection based on the amplitude and gyroscope values, and provides a web server to display the sensor readings and fall detection status in real-time.

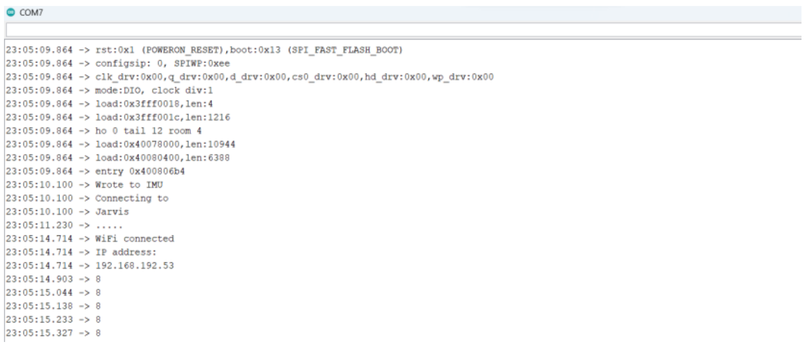
<u>Fall Detection System Integrated as a watch</u>





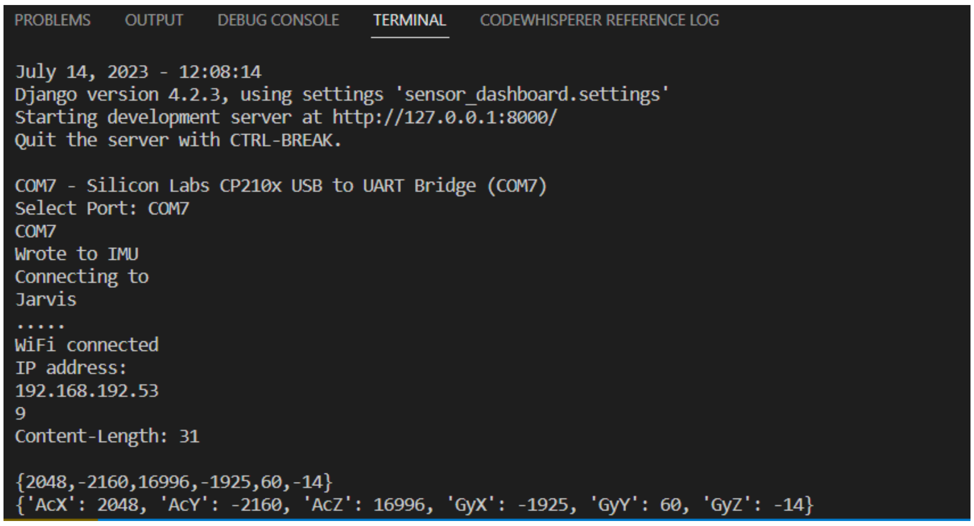
Result:

Serial monitor for wifi connection check:



```
23:05:09.864 -> rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
23:05:09.864 -> configsip: 0, SPIWP:0xee
23:05:09.864 -> clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
23:05:09.864 -> mode:DIO, clock div:1
23:05:09.864 -> load:0x3fff0018,len:4
23:05:09.864 -> load:0x3fff001c,len:1216
23:05:09.864 -> ho 0 tail 12 room 4
23:05:09.864 -> load:0x40078000,len:10944
23:05:09.864 -> load:0x40080400,len:6388
23:05:09.864 -> entry 0x400806b4
23:05:10.100 -> Wrote to IMU
23:05:10.100 -> Connecting to
23:05:10.100 -> Jarvis
23:05:11.230 -> .....
23:05:14.714 -> WiFi connected
23:05:14.714 -> IP address:
23:05:14.714 -> 192.168.192.53
23:05:14.903 -> 8
23:05:15.044 -> 8
23:05:15.138 -> 8
23:05:15.233 -> 8
23:05:15.327 -> 8
```

Serial Server output logs:

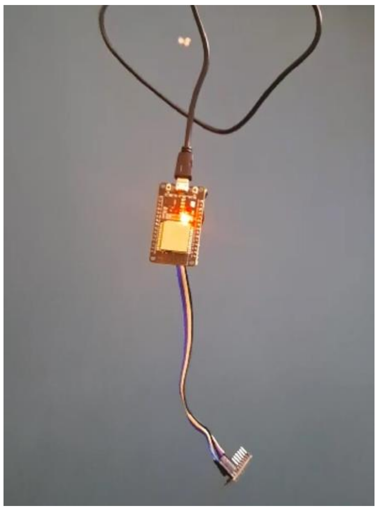


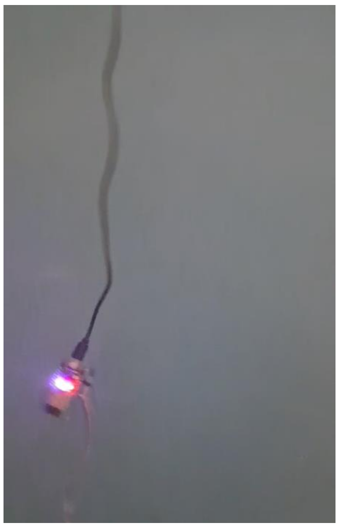
```
July 14, 2023 - 12:08:14
Django version 4.2.3, using settings 'sensor_dashboard.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.

COM7 - Silicon Labs CP210x USB to UART Bridge (COM7)
Select Port: COM7
COM7
Wrote to IMU
Connecting to
Jarvis
.....
WiFi connected
IP address:
192.168.192.53
9
Content-Length: 31

{2048,-2160,16996,-1925,60,-14}
{'AcX': 2048, 'AcY': -2160, 'AcZ': 16996, 'GyX': -1925, 'GyY': 60, 'GyZ': -14}
```

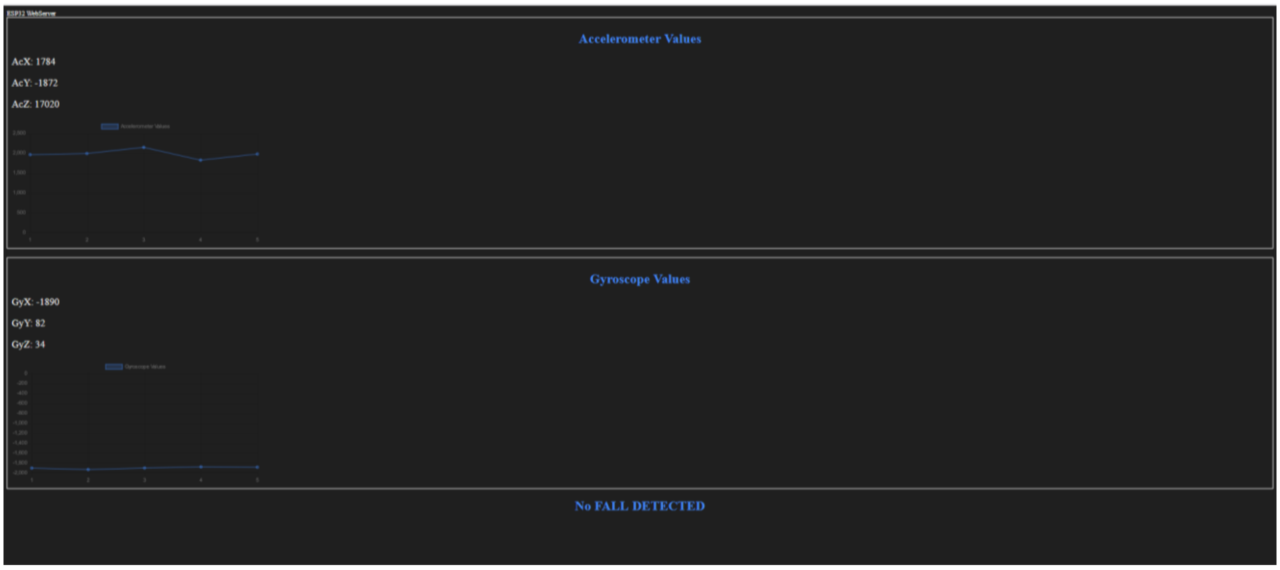
Under normal condition: **When Fall Detected:**





Web Application:

Using Django



ESP32 WebServer

Accelerometer Values

AcX: 1784

AcY: -1872

AcZ: 17020

Gyroscope Values

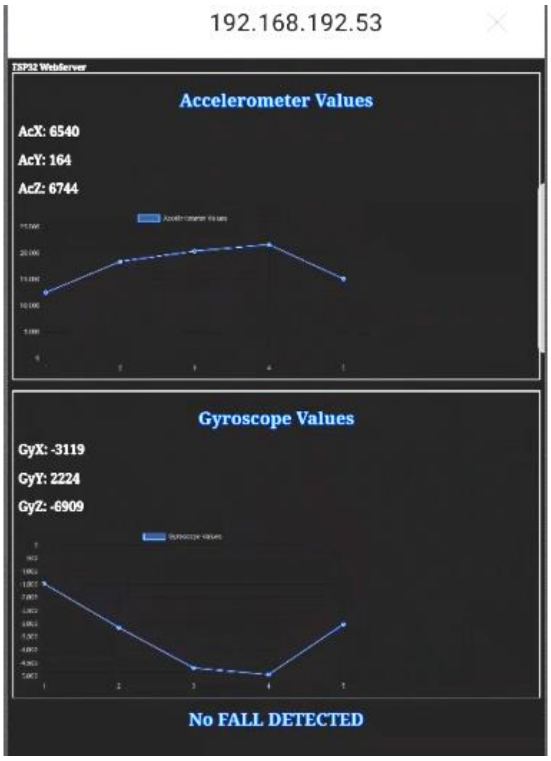
GyX: -1890

GyY: 82

GyZ: 34

No FALL DETECTED

Real-Time Data analysis in web server



192.168.192.53

ESP32 WebServer

Accelerometer Values

AcX: 6540

AcY: 164

AcZ: 6744

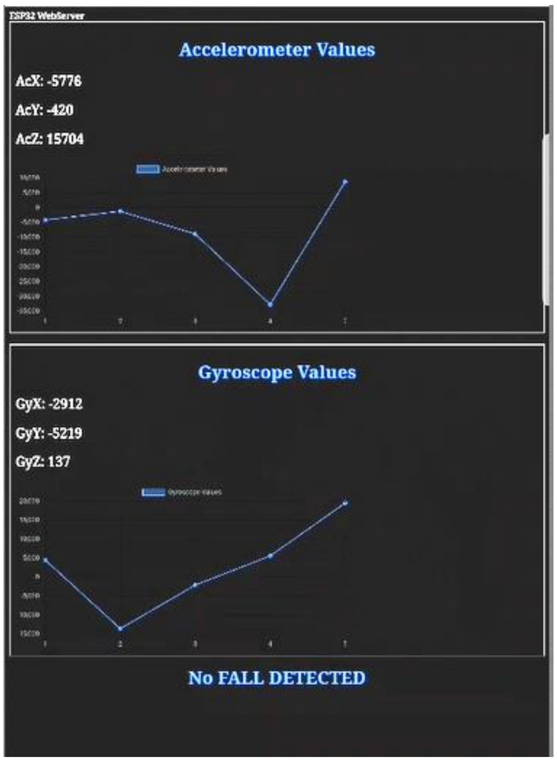
Gyroscope Values

GyX: -3119

GyY: 2224

GyZ: -6909

No FALL DETECTED



ESP32 WebServer

Accelerometer Values

AcX: -5776

AcY: -420

AcZ: 15704

Gyroscope Values

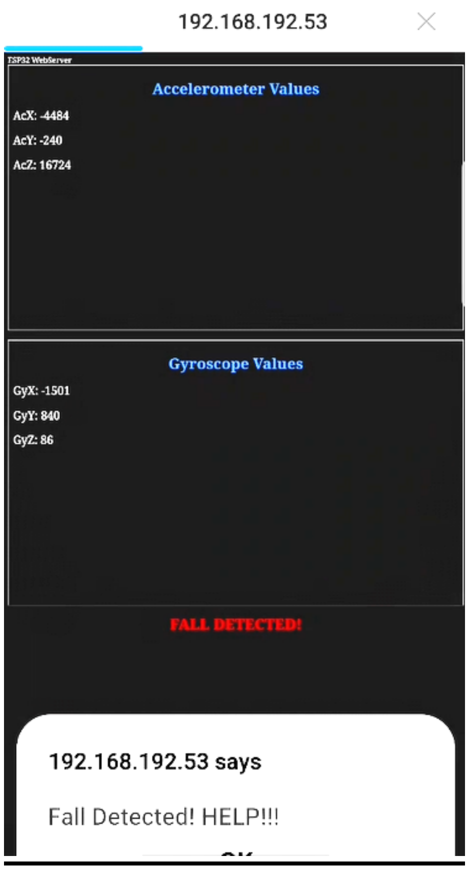
GyX: -2912

GyY: -5219

GyZ: 137

No FALL DETECTED

When Fall Event is Triggered:



192.168.192.53

ESP32 WebServer

Accelerometer Values

AcX: -4484

AcY: -240

AcZ: 16724

Gyroscope Values

GyX: -1501

GyY: 840

GyZ: 86

FALL DETECTED!

192.168.192.53 says

Fall Detected! HELP!!!

Conclusion:

The project consists of two building blocks, which are ESP32 microcontroller and IMU MPU6050 sensor. The project uses an advanced and highly accurate algorithms and formulae to detect a fall.

The proposed fall detection system has several advantages over traditional fall detection systems:

- It is more accurate and reliable than wearable sensors.
- It is more comfortable to wear.
- It is less prone to false alarms.
- It is more affordable.

The proposed fall detection system is a promising new technology that has the potential to reduce the risk of serious injury or death from falls in older adults. The system is accurate, reliable, comfortable to wear, and affordable. It is a valuable tool for caregivers and families of older adults.

The proposed system can be improved in the following ways:

- The system can be trained on a larger dataset of falls to improve its accuracy.
- The system can be tested in a variety of environments to improve its robustness.
- The system can be customized for different users to improve its usability.

Discussion:

- The objective of this project was to develop a fall detection system by interfacing the ESP32 microcontroller with the MPU6050 accelerometer and gyroscope sensor. The fall detection system aims to detect and alert caregivers or emergency services in the event of a fall, providing timely assistance to individuals at risk.
- The MPU6050 sensor was chosen for its ability to measure acceleration and angular velocity in all three axes. By continuously monitoring the sensor data, the ESP32 microcontroller can analyse changes in motion patterns and detect sudden falls.
- During the experimentation phase, the ESP32 was programmed using the Arduino IDE and the necessary libraries for MPU6050 sensor integration. The firmware code implemented an algorithm to analyse sensor readings and determine if a fall event occurred based on predefined thresholds and motion patterns associated with falling.
- To validate the system's performance, various simulated fall scenarios were created, including falls from different positions and angles. The ESP32 logged the sensor data and generated alerts whenever a fall was detected. These alerts were transmitted using various communication methods such as Wi-Fi.
- The accuracy and reliability of the fall detection system were evaluated by comparing the system's detection results with manual observations of the simulated fall scenarios. The system demonstrated high accuracy in detecting falls and successfully generated timely alerts.
- One of the limitations of the system was the need for proper sensor placement and calibration to ensure accurate readings, incorporation SMS/ Email alert system for the same. Factors such as sensor orientation and sensitivity adjustments affected the system's performance.
- In conclusion, this project successfully demonstrated the feasibility of developing a fall detection system by interfacing the ESP32 microcontroller with the MPU6050 sensor

which gave higher scope for learning IOT. The system exhibited reliable fall detection capabilities and has the potential to enhance the safety and well-being of individuals, particularly those prone to falling or at risk of injuries. Further development and fine-tuning of the system could improve its accuracy and expand its applications in healthcare and assisted living environments.

Further improvements could be made by incorporating machine learning algorithms to enhance the system's ability to differentiate falls from other activities.

References:

- N. Noury et al., "Fall detection - Principles and Methods," 2007 29th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, Lyon, France, 2007, pp. 1663-1666, doi: 10.1109/IEMBS.2007.4352627.
- Internet of Things (IoT) Fall Detection using Wearable Sensor, Loh Mei Yee et al 2019 J. Phys.: Conf. Ser. 1372 012048
- F. De Cillis, F. De Simio, F. Guido, R. A. Incalzi and R. Setola, "Fall-detection solution for mobile platforms using accelerometer and gyroscope data," 2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), Milan, Italy, 2015, pp. 3727-3730, doi: 10.1109/EMBC.2015.7319203.