VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY

UNIVERSITY OF INFORMATION TECHNOLOGY

FACULTY OF COMPUTER ENGINEERING

Full name: Đỗ Thanh Sơn

Student ID: 21522551

Class: CE224.O13.MTC

Full name: Trương Hữu Trường Sơn

Student ID: 21522559

Class: CE224.O13.MTC

CAPSTONE PROJECT

FALL DETECTION SYSTEM AND LOCATION AWARENESS

HỆ THỐNG PHÁT HIỆN NGÃ VÀ NHẬN BIẾT VỊ TRÍ

FINAL-TERM CAPSTONE PROJECT REPORT

MENTOR

PhD. TRI NHUT DO

VIETNAM NATIONAL UNIVERSITY
HO CHI MINH CITY
UIVERSITY OF INFORMATION
TECHNOLOGY

SOCIALIST REPUBLIC OF VIETNAM
Independence – Freedom – Happiness

FALL DETECTION SYSTEM AND LOCATION AWARENESS PROJECT

VIETNAMESE PROJECT NAME: Hệ thống phát hiện ngã và nhận biết vị trí
ENGLISH PROJECT NAME: Fall detection system and location awareness
Instructor PhD. Tri Nhut Do, Faculty of Computer Engineering
Implementation time: From: 1/10/2023 To: 1/11/2023
Student Perform: Truong Huu Truong Son – 21522559; Do Thanh Son - 21522551
Overview the topic: Fall detection system and location awareness involve developing systems and technologies to reliably identify falls and determine the precise location of individuals, with applications in safety monitoring. **The goal of the subject: The primary goal is to enhance safety and well-being by swiftly detecting falls and providing location information for timely assistance.** **Methods of implementation: Implementation methods include wearable sensors, GPS.** **Main contents of the topic: Key topics include sensor technologies, fall detection algorithms, data analysis, wearable devices, emergency response integration.**

Certification of Instructor	HCM City, 2023 November 19th Student

TABLE OF CONTENT

CHAPTER 1. INTRODUCTION

1.1 Abstract

Fall Detection System and Location Awareness are crucial for ensuring the safety of individuals in real-world scenarios. This project represents the actual implementation of a wearable device equipped with simple yet effective capabilities. It is designed to detect falls accurately and provide real-time location awareness. By focusing on monitoring these key motion parameters, the system ensures that individuals are safeguarded in their daily activities, making it a practical and tangible solution for enhancing personal safety.

1.2 Literature review

Numerous fall detection methodologies have been extensively researched and implemented across various facets of our daily lives. One prominent approach involves the integration of motion sensors, specifically accelerometers and gyroscopes. These sensors offer direct insights into both linear and angular motion, proving to be invaluable in the identification of genuine falls. Gyroscopes furnish valuable data regarding angular velocity, while accelerometers contribute vital information about linear motion. The amalgamation of these two sensor types significantly augments the accuracy of fall detection, surpassing the performance of a solitary accelerometer by more than 15%.

It is imperative to acknowledge that employing a greater number of sensors inevitably leads to increased power consumption. Consequently, there arises a challenge in designing algorithms adept at effectively fusing data from various sensor sources. Notably, for the specific task of human fall detection, sensors like the MPU6050 have proven to be sufficiently equipped for extracting essential information from their measurements.

In the context of this paper, a fall detection system is developed, centered around a wearable device. The hardware and software architecture of this device predominantly relies on the utilization of MPU6050, GPS NEO. Importantly, the device employs a resource-efficient and low-power consumption fall detection algorithm, rendering it a practical choice for indoor applications.

1.3 Purpose and Fuction

1.3.1 Purpose

The primary purpose of this project is to develop a wearable device that revolutionizes fall detection and location awareness. This device aims to address the limitations present in current systems by combining simplicity, accuracy, and real-time tracking capabilities. The purpose encompasses:

+ Enhancing Safety: By accurately identifying falls and providing immediate location data, the project strives to enhance the safety of individuals in indoor settings.
+ Prompt Emergency Response: The core goal is to ensure rapid response in emergency situations, minimizing the time between a fall occurrence and the arrival of assistance
+ Reliability and Accuracy: The system aims to minimize false alarms and accurately distinguish between regular movements and actual falls, improving overall reliability.
+ Accessibility: The project intends to create an affordable solution that can be accessible to a wide range of users, irrespective of economic backgrounds.

1.3.2 Fuction

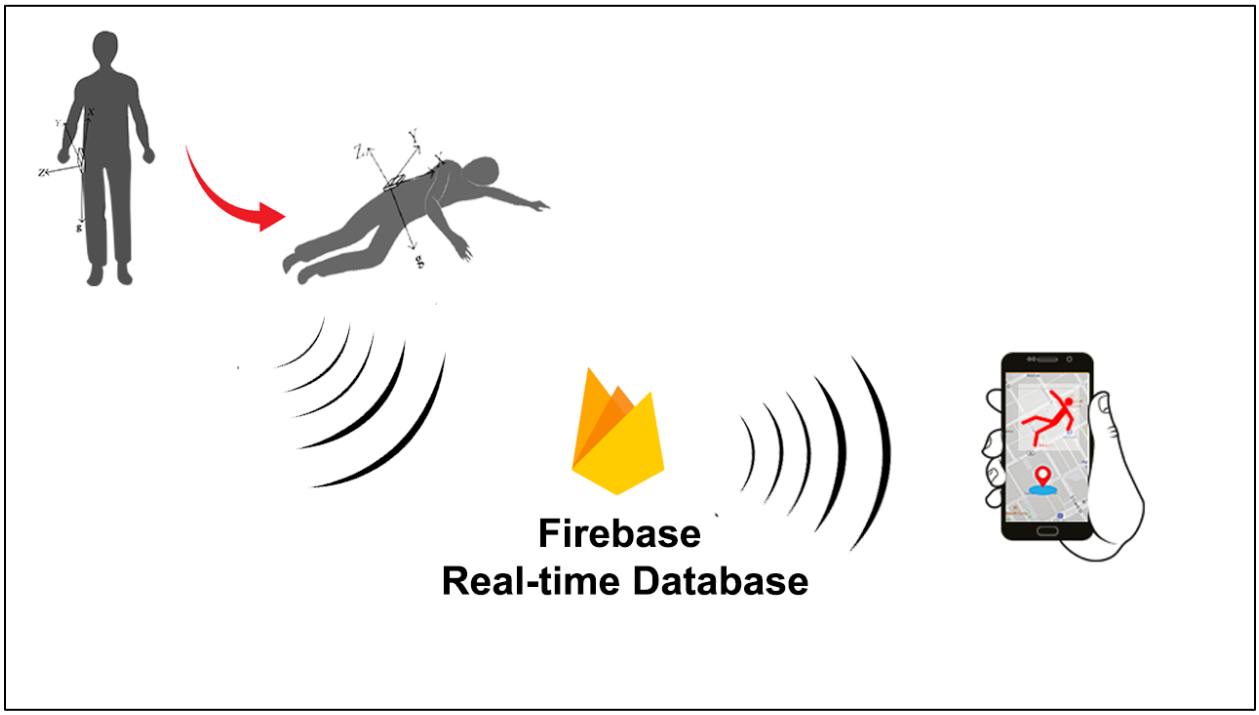


Figure 1. System architecture

The system's architecture, as illustrated in Figure 1, centers around a wearable device securely positioned at the waist of the user. This innovative system is adept at detecting falls. Upon detecting a fall event, it promptly ascertains the geographical location of individual and initiates the transmission of a concise fall alarm to an App . This rapid response mechanism ensures that individuals who have experienced a fall receive timely assistance, thereby minimizing any adverse consequences.

CHAPTER 2. THEORY BASIS

2.1 Wifi Manager Library

2.1.1 What is Wifi manager?

The WifiManager is a library for configuring ESP8266 to connect to a local WiFi network using a web interface. By initiating a dedicated WiFi network, ESP8266/NodeMCU allows other devices such as computers and smartphones to connect, redirecting all connections to the web interface created by ESP8266/NodeMCU. On this interface, users are provided with information to easily scan nearby networks, select a WiFi network, input a password, and save configurations.

2.1.2 How does wifimanger work?

```
#include <FS.h>          // this needs to be first, or it all crashes and burns...
#include <WiFiManager.h> // https://github.com/tzapu/WiFiManager

void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);
  Serial.println();
  WiFiManager wifiManager;

  //exit after config instead of connecting
  wifiManager.setBreakAfterConfig(true);

  //reset settings - for testing
  //wifiManager.resetSettings();

  if (!wifiManager.autoConnect("AutoConnectAP", "password")) {
    Serial.println("failed to connect, we should reset as see if it connects");
    delay(3000);
    ESP.restart();
    delay(5000);
  }
  Serial.println("connected...yeey :)");
}
```

Figure 2. Demonstration code for using Wifi Manager Library

When the ESP8266/NodeMCU boots up, by default, it will enter STATION mode and automatically connect to an Access Point using the saved connection information from the previous successful connection.

If the connection attempt fails (which could happen if the previous Access Point is no longer available, or if the password is incorrect, or if there is no saved information), the ESP8266/NodeMCU will enter AP mode. In this mode, it becomes an Access Point with its DNS pointing to itself and initiates a Web Server with the default address set to 192.168.4.1.

Devices that support Wi-Fi and have a web browser (such as smartphones, laptops, tablets, etc.) to connect to the newly created Access Point of the ESP8266/NodeMCU.

Typically, this will be a new Wi-Fi network with the name "Fall Detection Device," as shown in the following image:

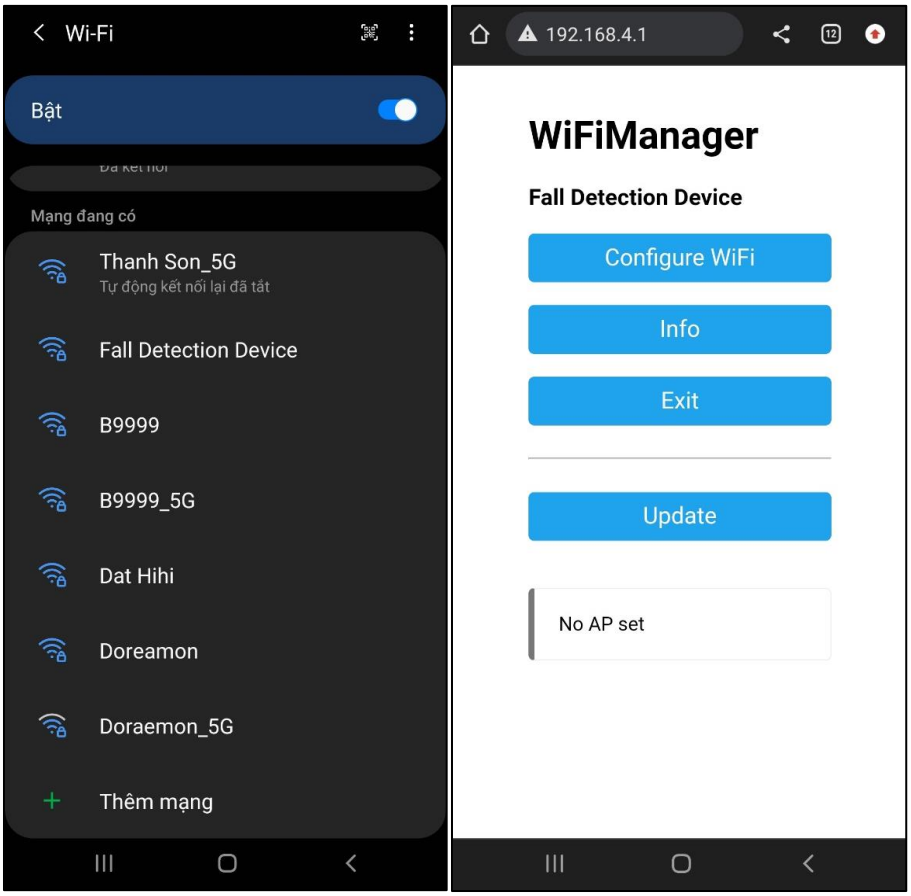


Figure 3. Setup wifi for "Fall detection Device"

2.2 GPS libraries

```ino
#include <SoftwareSerial.h>
#include <TinyGPS++.h>

// Define the RX and TX pins for SoftwareSerial
const int rxPin = 13;
const int txPin = 15;

// Create a SoftwareSerial object
SoftwareSerial gpsSerial(rxPin, txPin);

// Create a TinyGPS++ object
TinyGPSPlus gps;

void setup() {
  Serial.begin(9600);  // Serial communication for debugging
  gpsSerial.begin(9600);  // SoftwareSerial communication with GPS module
}

void loop() {
  // Read data from the GPS module
  while (gpsSerial.available() > 0) {
    if (gps.encode(gpsSerial.read())) {
      // If new GPS data is available, print the information
      if (gps.location.isUpdated()) {
        Serial.print("Latitude: ");
        Serial.println(gps.location.lat(), 6);

        Serial.print("Longitude: ");
        Serial.println(gps.location.lng(), 6);

        Serial.print("Altitude: ");
        Serial.println(gps.altitude.meters());
      }
    }
  }
```

Figure 4. Demonstration code for using SoftwareSerial and TinyGPS libraries

SoftwareSerial.h is a library in Arduino that allows you to create a software-based serial communication port on any of the digital pins of the Arduino board. This is useful when you need additional serial ports for communication with other devices, as not all Arduino boards have multiple hardware serial ports.

TinyGPSPlus.h is a library designed for parsing NMEA data from GPS modules. It simplifies the process of extracting relevant information such as latitude, longitude, altitude, and more from the raw data received from GPS modules. It provides an easy-to-use interface for working with GPS data.

2.3 Geographic coordinates

Latitude: Latitude specifies the north-south position of a point on the Earth's surface. It is measured in degrees, with the equator at 0 degrees latitude and the poles at 90 degrees north and south latitude, respectively.

Longitude: Longitude indicates the east-west position of a point on the Earth's surface. It is also measured in degrees, with the prime meridian at 0 degrees longitude and values ranging up to 180 degrees east or west.

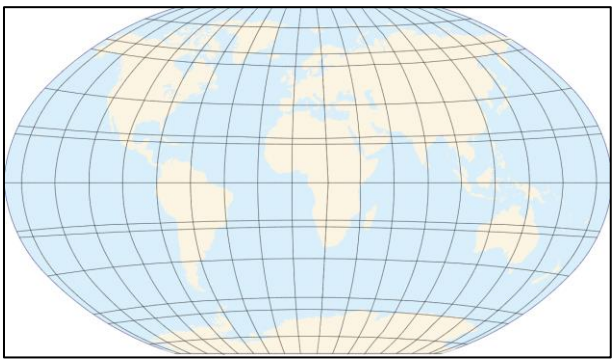


Figure 5. Demonstration of Latitude and Longtitude

Altitude: Altitude, also known as elevation or height above sea level, represents the vertical distance of a point on the Earth's surface above or below a reference point, usually mean sea level. It is measured in meters or feet.

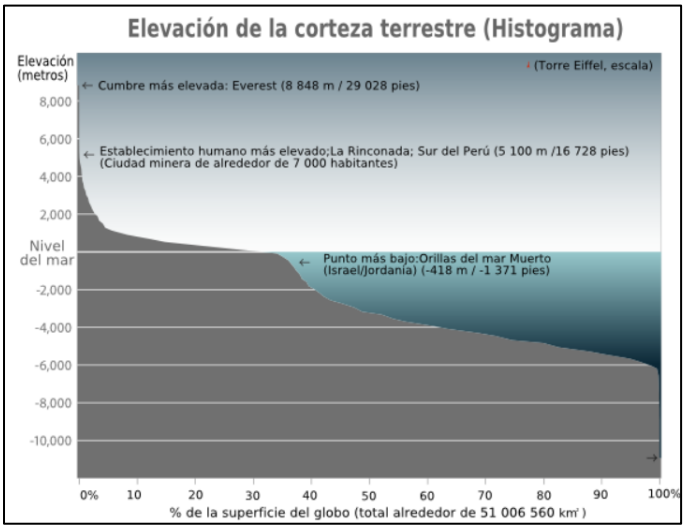


Figure 6. Demonstration of Altitude

2.4 Mobile app development with MIT App Inventor

2.4.1 Development Tool: MIT App Inventor

MIT App Inventor is a web-based tool for developing Android mobile applications that allows users to create apps without requiring deep programming skills. With a visual interface and the ability to drag and drop "blocks" to build functionality, App Inventor is suitable for beginners in mobile app development and is widely used in education. This tool supports various features, including sensors, network connectivity, and interaction with databases.

2.4.2 App Functionality

The fall detection mobile application comprises three main sections, accessible through a sliding sidebar:

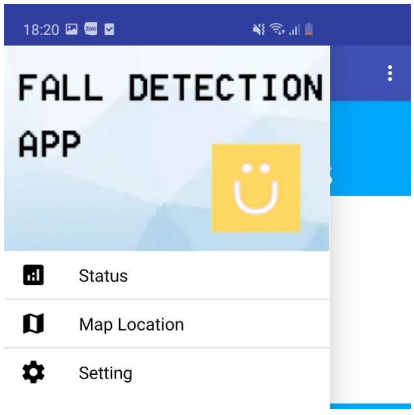
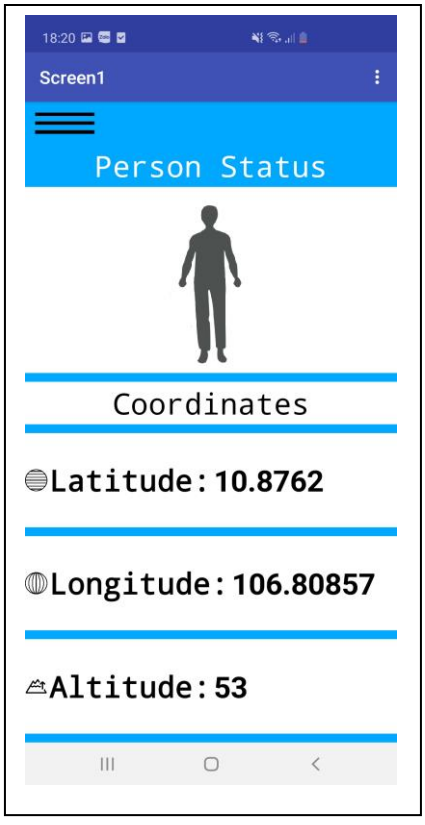
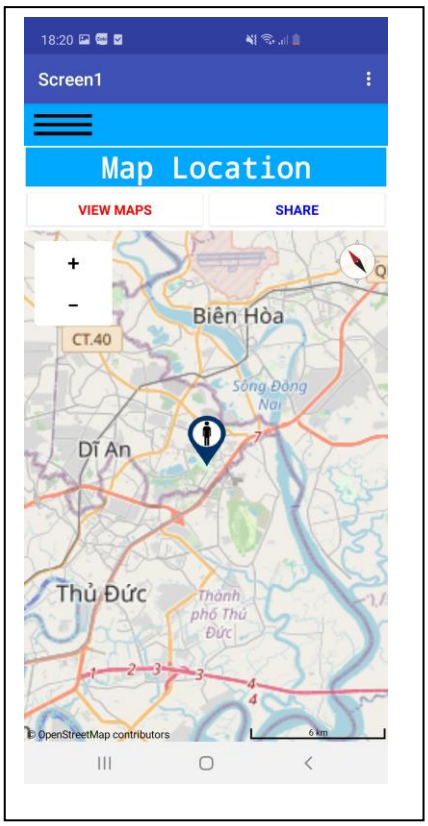


Figure 7. Slidebar on mobile app

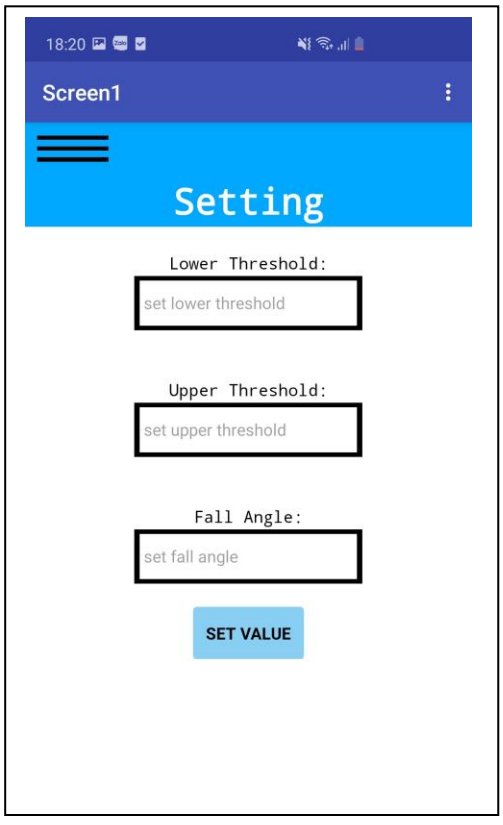
Slidebar contents contain:
- Status Section:
User Status: Indicates whether the user is currently standing or has experienced a fall.
Coordinates: Displays three crucial geographical coordinates - longitude, latitude, and altitude - providing the precise location of the user.
- Map Location Section:
Marker Display: Illustrates the user's current location with a marker on the map.
Google Map View Button: Enables users to view their location on Google Maps directly from the application.
Coordinate Sharing Button: Allows users to share their coordinates with multiple applications for additional functionalities.
- Setting Section:
Threshold Setup: Provides the option to set a threshold for fall detection, allowing users to customize the sensitivity of the system.
Angle Fall Setup: Allows users to configure the angle of fall for the device, enhancing the adaptability of the fall detection mechanism.



a) Status section



b) Map location section



c) Setting section

Source App:
https://drive.google.com/file/d/1xB_vrccJUdR_U2U-WW3wS0urMfiUT4-Y/view?usp=sharing

2.5 Real-time Database Integration with Firebase

2.5.1 What is Firebase Realtime Database?

Firebase Realtime Database is a part of the Firebase suite, a set of development platforms and cloud computing services offered by Google. Specifically, the Realtime Database is a cloud-hosted NoSQL database that allows developers to store and synchronize data in real-time across multiple clients. It is well-integrated with various platforms and languages, including Android, iOS, JavaScript, Node.js, Java, Unity, PHP, and C++.

2.5.2 Firebase Realtime Database Functionality

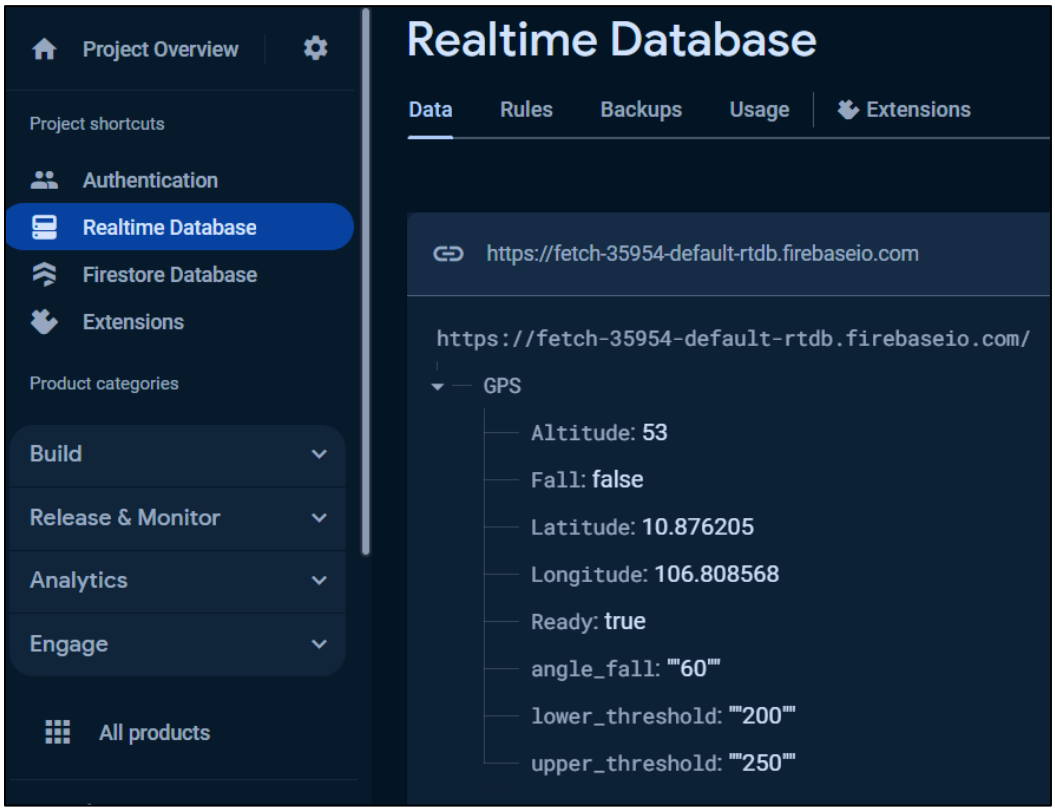


Figure 8. Firebase Realtime Database

The Realtime Database for this fall detection system is structured into a single group, encompassing essential parameters:

/gps

/altitude: Records the altitude information.

/Fall: Indicates the fall event, providing real-time updates.

/latitude: Stores the latitude coordinates of the user's location.

/longitude: Stores the longitude coordinates of the user's location.

/angle_fall: Set the fall angle of the fall event.

/lower_threshold: Sets the lower threshold for fall detection sensitivity.

/upper_threshold: Establishes the upper threshold for fall detection sensitivity.

By utilizing Firebase Realtime Database, the fall detection system can seamlessly store, update, and retrieve critical data in real-time, facilitating instantaneous communication between the wearable device and the associated mobile application. This integration enhances the overall responsiveness and reliability of the fall detection system, ensuring timely and accurate transmission of crucial information.

2.6 Accelerometer and Gyroscope

2.6.1 Accelerometer

An accelerometer is a vital sensor employed to quantify acceleration, a parameter defined as the speed at which an object accelerates or decelerates while moving in a specific direction. In its physical form, an accelerometer is crafted as a microelectromechanical system (MEMS) device, meticulously manufactured through microfabrication technology. This intricate process involves a multilayer wafer fabrication, where the accelerometer gauges acceleration forces by discerning the displacement of a mass concerning stationary electrodes. Essentially, accelerometers play a crucial role in diverse applications, ranging from smartphones and wearable fitness devices to automotive safety systems, providing valuable data by precisely capturing changes in velocity and movement.

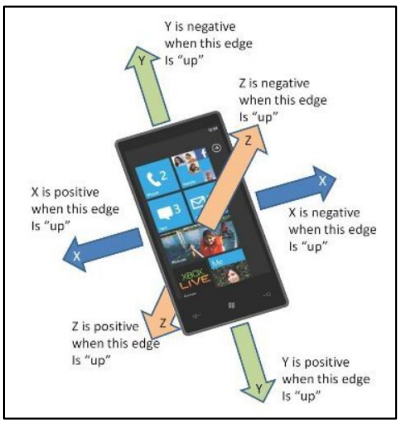


Figure 9. Demonstration of the axis of Accelerometer

2.6.2 Gyroscope

A gyroscope is a device designed to apply the principles of conserving angular momentum for the purpose of measuring or maintaining orientation. This sensor operates by utilizing the concept of angular momentum conservation, enabling accurate detection and tracking of changes in orientation. Gyroscopes find widespread applications in various fields, from navigation systems in aircraft and spacecraft to stabilization mechanisms in electronic devices, playing a crucial role in preserving and determining directional information.

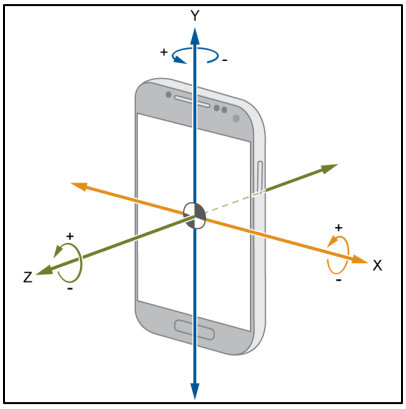


Figure 10. Demonstration of the axis of Gyroscope

CHAPTER 3. SYSTEM DESIGN

3.1 Project layout

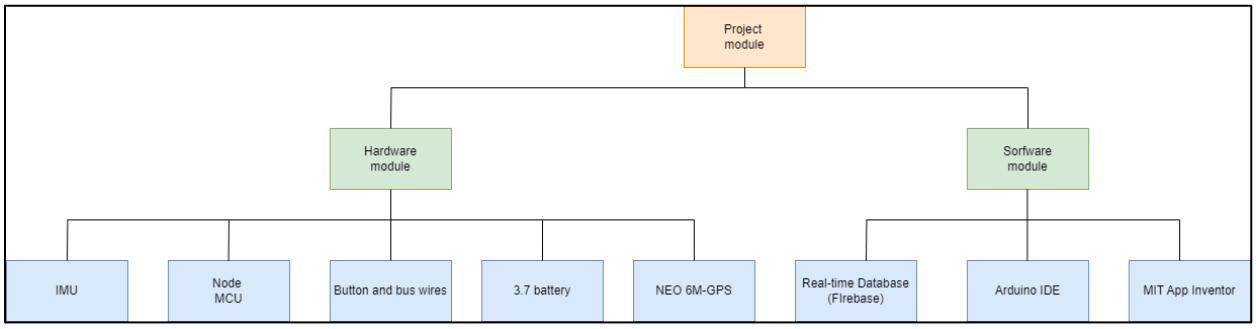


Figure 11. Layout of project module

Node MCU (ESP8266): Node MCU is the microcontroller unit in the prototype. It has an inbuilt Wi-Fi module (ESP8266) that establishes wireless remote switching of home appliances.

IMU (Inertial Measurement Unit): The IMU, utilizing components such as the MPU6050, detects and monitors the user's movement and orientation to identify potential falls in real-time.

Button and Bus Wires: The button, along with bus wires, provides manual control or input for the user to withdraw the action of falling, interrupting the fall detection process if necessary.

Neo-6M GPS: The Neo-6M GPS module retrieves real-time geographical coordinates, enabling accurate location tracking and data about the user's position.

Real-time Database (Firebase): The Real-time Database, powered by Firebase, serves as the central repository for storing, retrieving, and managing data related to fall detection events, user coordinates, and status.

Arduino IDE: The Arduino Integrated Development Environment (IDE) is utilized for writing, compiling, and uploading code to the Node MCU (ESP8266) to manage the functionality of the prototype and the interaction between the various hardware components.

MIT App Inventor: The MIT App Inventor is employed to create a user-friendly smartphone application, allowing users or caregivers to access and visualize the data from the Real-time Database (Firebase), providing insights into the user's fall status and coordinates in real-time.

3.2 Hardware Design

3.2.1 Block Diagram

a) Node MCU

NodeMCU (Node Microcontroller Unit) is a low-cost open source IOT platform. It initially included firmware which runs on the ESP8266 Wi-Fi SoC from Espressif Systems, and hardware which was based on the ESP-12 module. Later, support for the ESP32 32-bit MCU was added.

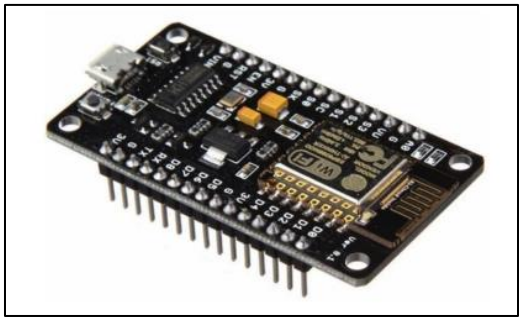


Figure 12. Node MCU Development Board.

NodeMCU is an open source firmware for which open source prototyping board designs are available. The name “NodeMCU” combines “node” and “MCU” (micro-controller unit). The term “NodeMCU” strictly speaking refers to the firmware rather than the associated development kits. Both the firmware and prototyping board designs are open source. The firmware uses the Lua scripting language. The firmware is based on the eLua project, and built on the Espressif Non-OS SDK for ESP8266. It uses many open source projects, such as luacjson and SPIFFS. Due to resource constraints, users need to select the modules relevant for their project and build a firmware tailored to their needs. Support for the 32-bit ESP32 has also been implemented.

The prototyping hardware typically used is a circuit board functioning as a dual in-line package (DIP) which integrates a USB controller with a smaller surface-mounted board containing the MCU and antenna. The choice of the DIP format allows for easy prototyping on breadboards. The design was initially was based on the ESP-12 module of the ESP8266, which is a Wi-Fi SoC integrated with a Tensilica Xtensa LX106 core, widely used in IOT applications.

b) NEO-6M GPS Module

The NEO-6M is a widely used and relatively low-cost GPS (Global Positioning System) module. It is commonly utilized in a variety of applications where accurate positioning and location data are required.

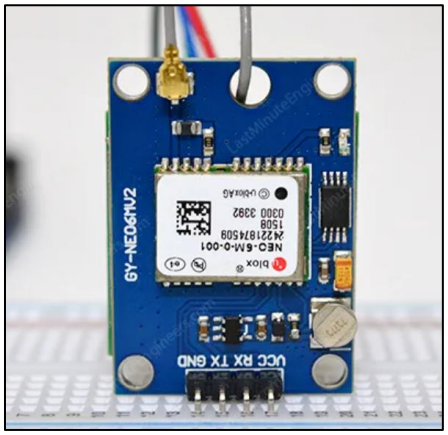


Figure 13. NEO-6M GPS Module

The NEO-6M GPS module is a versatile and cost-effective solution for adding GPS capabilities to a wide range of projects, including vehicle tracking systems, location-based services, geocaching devices, drones, and more. Its ease of integration, accuracy, and fast time to first fix make it a popular choice for many GPS-related applications.

c) MPU6050 Module

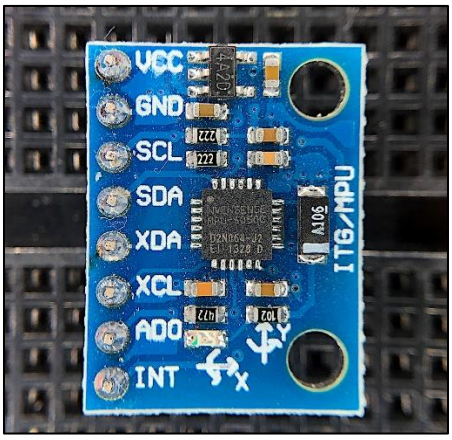


Figure 14. MPU6050 Module

A highly well-liked accelerometer gyroscope chip with six sensed axes and a 16-bit measurement resolution is the MPU6050. A gyroscope and an accelerometer are used to create an inertial measurement unit, or IMU. IMU sensors are used in many different types of devices, including smartphones, tablets, satellites, spacecraft, drones, unmanned aerial vehicles (UAVs), robots, and many more. They are used for aircraft control, orientation and position sensing, motion tracking, and other functions.

d) Hardware structure

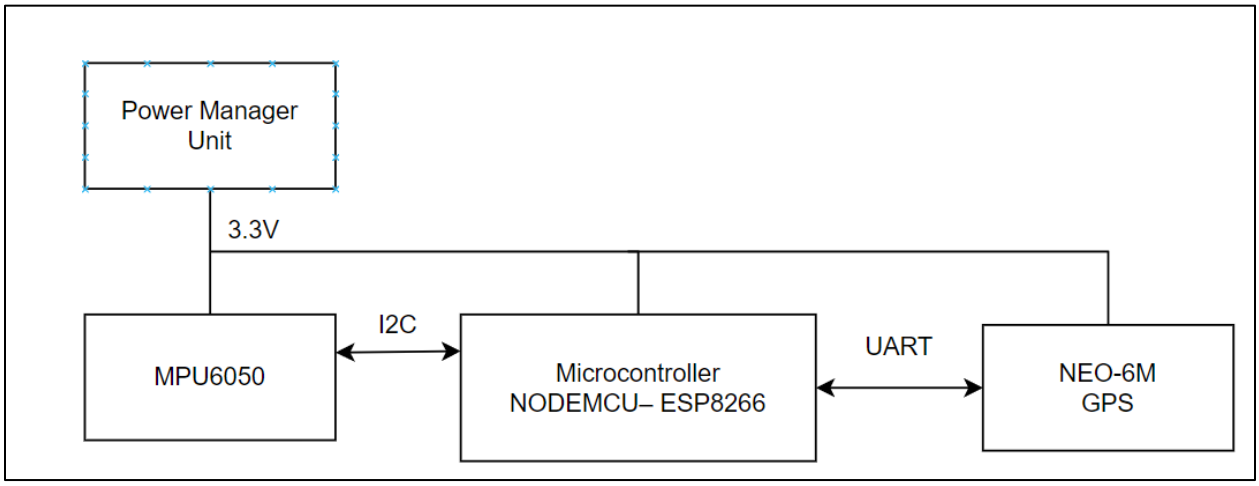


Figure 15. Basic hardware structure

The NodeMCU serves as the control unit, collecting data from the MPU6050 and GPS module, and when a fall is detected, it can send an alert to mobile app through wifi to notify relevant parties. The MPU6050's motion data and the NEO-6M GPS data are crucial for detecting a fall event accurately.

3.2.2 Circuit diagram

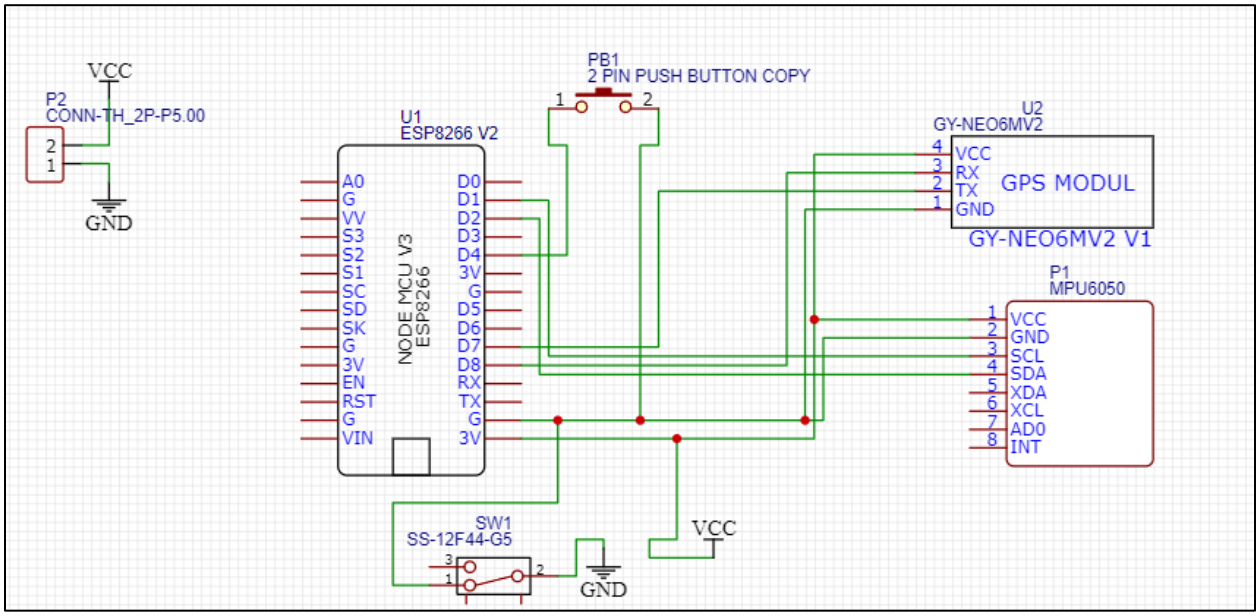


Figure 16. 3D Schematic of Fall Detection Device

3.3 Sorfware Design

3.2.1 Flowchart

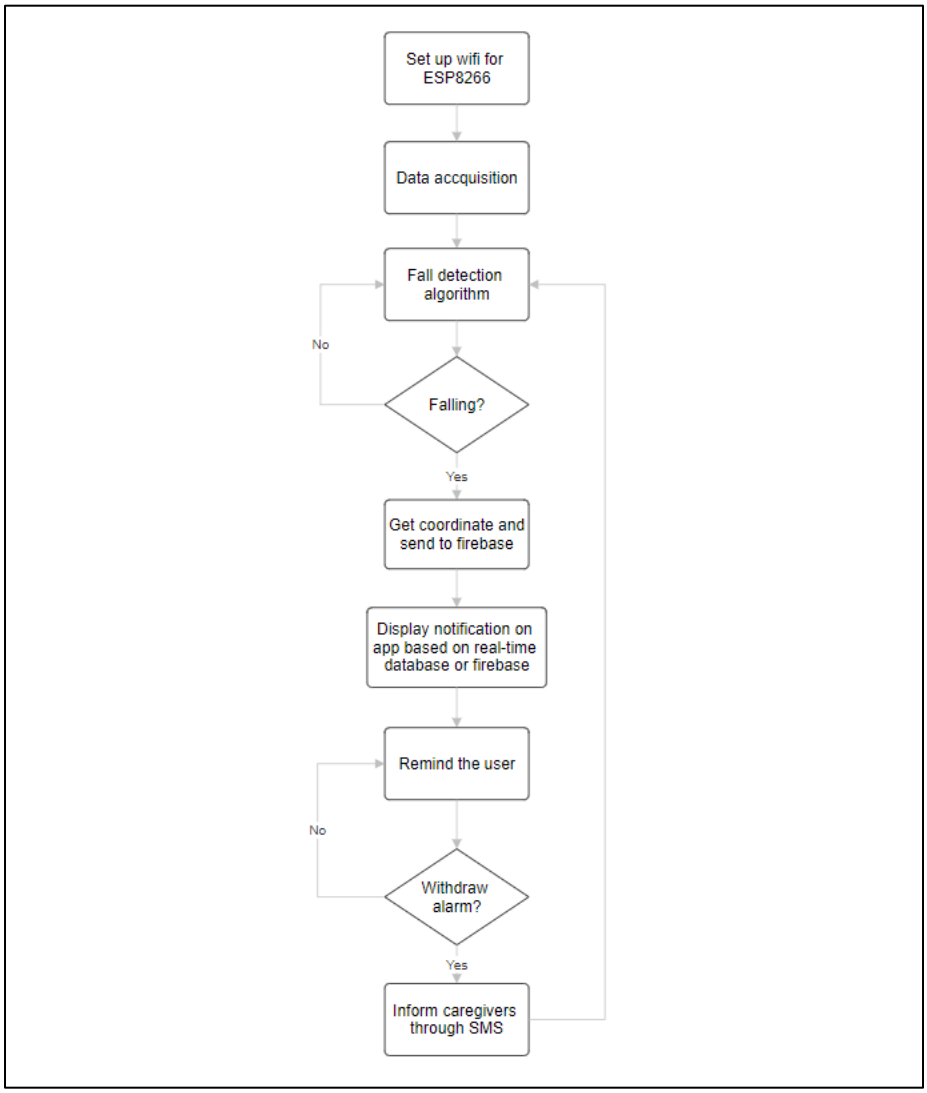


Figure 17. Flow chart

3.2.1.1 Flow chart of fall detection algorithm

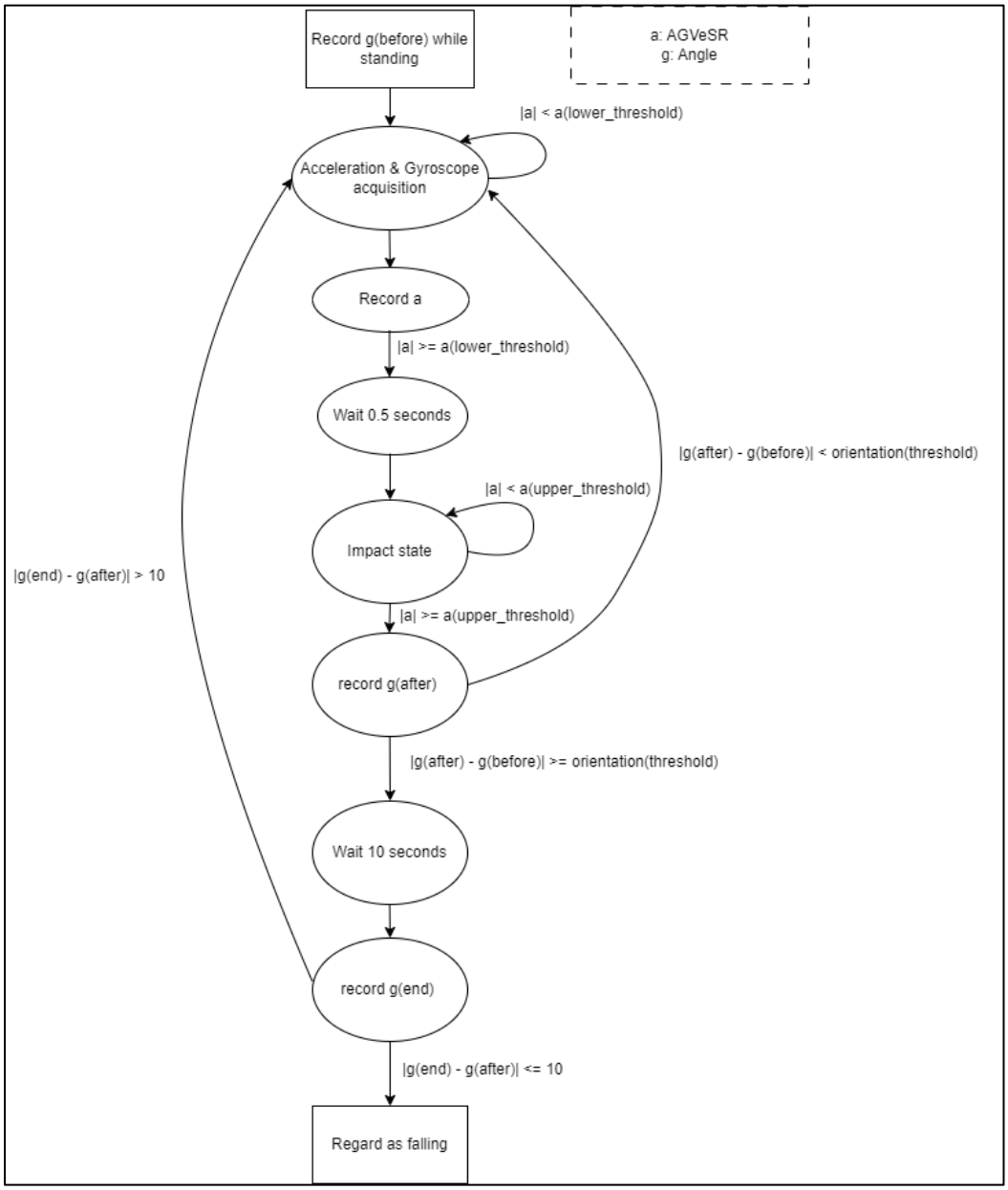


Figure 18. Flow chart of fall detection algorithm

3.2.2 Algorithm

a) *Data accquisition and calibration*

- Gyroscope initial:

```
void Read_MPU(void) {
  Wire.beginTransmission(0x68);
  Wire.write(0x1A);
  Wire.write(0x05);
  Wire.endTransmission();
  Wire.beginTransmission(0x68);
  Wire.write(0x1B);
  Wire.write(0x8);
  Wire.endTransmission();
  Wire.beginTransmission(0x68);
  Wire.write(0x43);
  Wire.endTransmission();
  Wire.requestFrom(0x68,6);
  int16_t RatePitchroX=Wire.read()<<8 | Wire.read();
  int16_t RatePitchroY=Wire.read()<<8 | Wire.read();
  int16_t RatePitchroZ=Wire.read()<<8 | Wire.read();
```

Figure 19. Reading signals gyroscope from MPU6050

+ Explanation:

(72-75): Switch on low pass filter

4.3 Register 26 – Configuration
CONFIG

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1A	26	-	-	EXT_SYNC_SET[2:0]			DLPF_CFG[2:0]		

Description:

This register configures the external Frame Synchronization (FSYNC) pin sampling and the Digital Low Pass Filter (DLPF) setting for both the gyroscopes and accelerometers.

The DLPF is configured by *DLPF_CFG*. The accelerometer and gyroscope are filtered according to the value of *DLPF_CFG* as shown in the table below.

DLPF_CFG	Accelerometer (F_s = 1kHz)		Gyroscope		
	Bandwidth (Hz)	Delay (ms)	Bandwidth (Hz)	Delay (ms)	Fs (kHz)
0	260	0	256	0.98	8
1	184	2.0	188	1.9	1
2	94	3.0	98	2.8	1
3	44	4.9	42	4.8	1
4	21	8.5	20	8.3	1
5	10	13.8	10	13.4	1
6	5	19.0	5	18.6	1
7	RESERVED		RESERVED		8

Figure 20. Low pass filter accorrding to MPU6050 datasheet

(76-79): Set the sensitivity scale factor

(76-79): Set the sensitivity scale factor(+-500 deg/s)

4.4 Register 27 – Gyroscope Configuration
GYRO_CONFIG

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1B	27	XG_ST	YG_ST	ZG_ST	FS_SEL[1:0]		-	-	-

Description:

This register is used to trigger gyroscope self-test and configure the gyroscopes' full scale range.

Each 16-bit gyroscope measurement has a full scale defined in *FS_SEL* (Register 27). For each full scale setting, the gyroscopes' sensitivity per LSB in *GYRO_xOUT* is shown in the table below:

FS_SEL	Full Scale Range	LSB Sensitivity
0	± 250 °/s	131 LSB/°/s
1	± 500 °/s	65.5 LSB/°/s
2	± 1000 °/s	32.8 LSB/°/s
3	± 2000 °/s	16.4 LSB/°/s

Figure 21. Set the sensitivity scale of gyroscope.

(80-82) : Access registers storing gyro measurements

4.19 Registers 67 to 72 – Gyroscope Measurements
GYRO_XOUT_H, GYRO_XOUT_L, GYRO_YOUT_H, GYRO_YOUT_L, GYRO_ZOUT_H, and GYRO_ZOUT_L

Type: Read Only

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
43	67	GYRO_XOUT[15:8]							
44	68	GYRO_XOUT[7:0]							
45	69	GYRO_YOUT[15:8]							
46	70	GYRO_YOUT[7:0]							
47	71	GYRO_ZOUT[15:8]							
48	72	GYRO_ZOUT[7:0]							

Figure 22. Access register storring gyroscope measurements

(83-86): Read the gyro measurements around the X axis

```
 98    RateRoll=(float)RatePitchroX/65.5;
 99    RatePitch=(float)RatePitchroY/65.5;
100    RateYaw=(float)RatePitchroZ/65.5;
```

Figure 23. Gyroscope measurements around the x axis

(98-100): Convert the measurements units to degree/s

```
  Wire.setClock(400000);
  Wire.begin();
  delay(250);
  Wire.beginTransmission(0x68);
  Wire.write(0x6B);
  Wire.write(0x00);
  Wire.endTransmission();
  for (RateCalibrationNumber=0; RateCalibrationNumber<2000; RateCalibrationNumber ++) {
    Read_MPU();
    RateCalibrationRoll+=RateRoll;
    RateCalibrationPitch+=RatePitch;
    RateCalibrationYaw+=RateYaw;
    delay(1);
  }
  RateCalibrationRoll/=2000;
  RateCalibrationPitch/=2000;
  RateCalibrationYaw/=2000;
  RateRoll -= RateCalibrationRoll;
  RatePitch -= RateCalibrationPitch;
  RateYaw -= RateCalibrationYaw;
}
```

Figure 24. Setup clock and calibration of gyroscope

(145): Set the clock speed of I2C

The MPU-6000 family is comprised of two parts, the MPU-6000 and MPU-6050. These parts are identical to each other with two exceptions. The MPU-6050 supports I^2C communications at up to 400kHz and has a VLOGIC pin that defines its interface voltage levels; the MPU-6000 supports SPI at up to 20MHz in addition to I^2C, and has a single supply pin, VDD, which is both the device's logic reference supply and the analog supply for the part.

Figure 25. Clock speed according to MPU6050 Datasheet

(149-151) Set to zero (wakes up the MPU-6050)

(152-164) Perform the calibration measurements aand caculate the calibration values

```
// FALL ALGORITHM
// Gyroscope calibration
Read_MPU();
RateRoll -= RateCalibrationRoll;
RatePitch -= RateCalibrationPitch;
RateYaw -= RateCalibrationYaw;
```

Figure 26. Gyroscope values after calibration

(250-253) Gyroscope values after calibration

- Accelerometer initial:

```
Wire.beginTransmission(0x68);
Wire.write(0x1C);
Wire.write(0x10);
Wire.endTransmission();
Wire.beginTransmission(0x68);
Wire.write(0x3B);
Wire.endTransmission();
Wire.requestFrom(0x68,6);
int16_t AccXLSB = Wire.read() << 8 | Wire.read();
int16_t AccYLSB = Wire.read() << 8 | Wire.read();
int16_t AccZLSB = Wire.read() << 8 | Wire.read();
```

Figure 27. Reading signals accelerometer from MPU6050

+ Explanation:

(87-90): Configure the accelerometer output (+-8G).

4.5 Register 28 – Accelerometer Configuration
ACCEL_CONFIG

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1C	28	XA_ST	YA_ST	ZA_ST	AFS_SEL[1:0]		-		

Description:

This register is used to trigger accelerometer self test and configure the accelerometer full scale range. This register also configures the Digital High Pass Filter (DHPF).

AFS_SEL selects the full scale range of the accelerometer outputs according to the following table.

AFS_SEL	Full Scale Range
0	± 2*g*
1	± 4*g*
2	± 8*g*
3	± 16*g*

Figure 28. Set accelerometer scale range

(91-97): Pull the accelerometer measurements from the sensor.

4.17 Registers 59 to 64 – Accelerometer Measurements
ACCEL_XOUT_H, ACCEL_XOUT_L, ACCEL_YOUT_H, ACCEL_YOUT_L, ACCEL_ZOUT_H, and ACCEL_ZOUT_L

Type: Read Only

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
3B	59	ACCEL_XOUT[15:8]							
3C	60	ACCEL_XOUT[7:0]							
3D	61	ACCEL_YOUT[15:8]							
3E	62	ACCEL_YOUT[7:0]							
3F	63	ACCEL_ZOUT[15:8]							
40	64	ACCEL_ZOUT[7:0]							

Figure 29 Access accelerometer registers

```
101   AccX=(float)AccXLSB/4096-0.01;
102   AccY=(float)AccYLSB/4096+0.01;
103   AccZ=(float)AccZLSB/4096-0.06;
```

Figure 30. Accelomete values after calibration

(101-103): Convert the measuremnets to physical values and remove off-set

b) *Record g(before) as standing position*

```
// Calculate the angle after 1-second intervals
if (count <= 250) {
  angle_before = Alpha_Degree();
  count++;
}
if (count == 250) {
  Serial.println("Accelerometer range set to: +-8G");
  Serial.println("Gyro range set to: +-500 deg/s");
  Serial.println("----Based Angle Calculated----");
}
```

Figure 31. Caculate based angle for one second intervals.

- Explanation:

The function Alpha_Degree() computes the roll angle using a one-dimensional Kalman filter. The filter processes the gyroscope data (RateRoll) and accelerometer data (AngleRoll) to estimate and refine the roll angle, providing a more accurate measurement in KalmanAngleRoll. The uncertainty or variance in this angle estimation is stored in KalmanUncertaintyAngleRoll.

Within the loop(), the angle is calculated at 1-second intervals (count <= 250). The Alpha_Degree() function is called to compute the angle, and the count is incremented. Once the count reaches 250, the program prints messages to the serial monitor indicating the setup for accelerometer and gyroscope ranges, and it announces the calculation of the angle based on the provided data.
Utilizing a Kalman filter to refine the angle estimation over time.

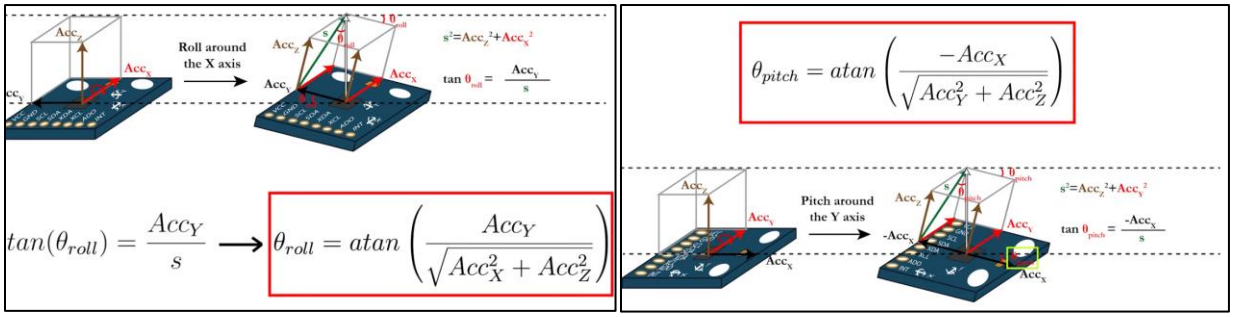
c) Compute Angle Roll using Kalman Filter

- Caculate Angle Roll using accelerometer:

This formula utilizes accelerometer data to estimate the roll and pitch angles. It leverages the relationships between the different axes of the accelerometer (typically measuring acceleration due to gravity) to determine the device's orientation relative to the ground.

For the roll angle calculation (AngleRoll), the formula uses the measurements along the Y-axis (representing the device's side-to-side movement) and factors in the square root of the sum of squares of the X and Z-axis values. By applying the arctangent function, it computes the angle necessary to align the Y-axis with the gravity vector, essentially providing the angle at which the device is tilted sideways.

Similarly, for the pitch angle (AnglePitch), it involves the X-axis (representing the device's front-to-back movement) and uses the Y and Z-axis measurements. The formula determines the angle required to align the X-axis with the direction of gravity, indicating the forward or backward tilt of the device.



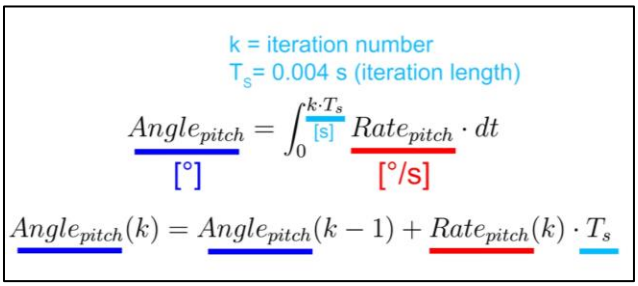

```
AngleRoll = atan(AccY / sqrt(AccX * AccX + AccZ * AccZ)) * 180 / M_PI;
AnglePitch = -atan(AccX / sqrt(AccY * AccY + AccZ * AccZ)) * 180 / M_PI;
```

Figure 32. Fomula for caculating Angle Roll (accelerometer)

- Caculate Roll Rate using gyroscope:

A 250 Hertz loop, the method for calculating the current angle based on the rotation rate and the time interval. The formula involves summing the rotation rates multiplied by the iteration time (in this case, 4 milliseconds) to the angle from the previous iteration to obtain the angle for the current iteration. However, the passage of measurement errors from previous iterations introduces a rapid drift in the calculated angle, impacting its accuracy and reliability.

Moreover, the formula highlights a crucial problem: the gyroscope-based approach isn't immune to changes in angles during movement.



```
// FALL ALGORITHM
// Gyroscope calibration
Read_MPU();
RateRoll -= RateCalibrationRoll;
RatePitch -= RateCalibrationPitch;
RateYaw -= RateCalibrationYaw;
```

Figure 33. Fomula for caculating Roll Rate(gyroscope)

- Kalman Filter 1-Dimensional:

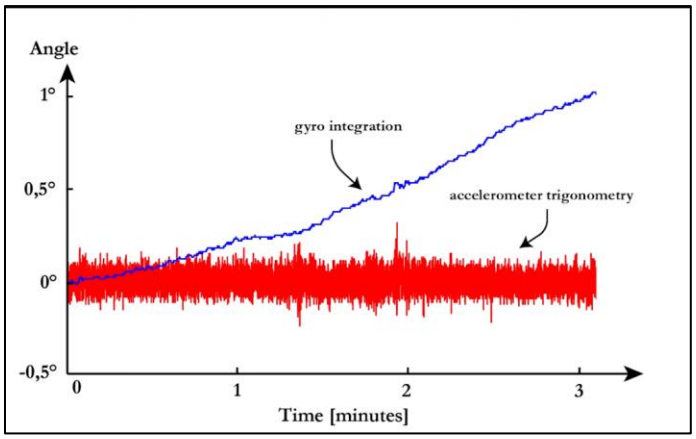


Figure 34. Comparision Roll Angle(accelerometer) and RateRoll(gyroscope)

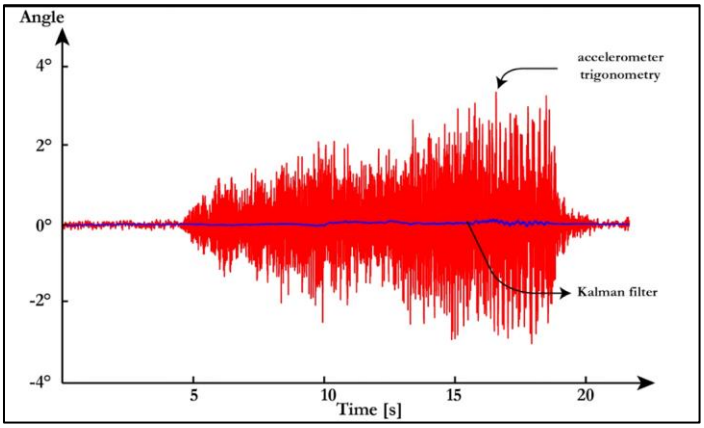


Figure 35. Comparision Roll Angle (Kalman Filter) and Roll Angle(accelerometer)

- Reason why we use Kalman Filter:

Employing a one-dimensional Kalman filter for fall detection offers a precise and effective method for estimating body orientation crucial in identifying potential falls. The filter's one-dimensional implementation simplifies the fusion of sensor data, such as accelerometer or gyroscope measurements, focusing specifically on the angle of orientation. This simplified model reduces computational complexity while effectively handling noisy sensor data, enhancing the accuracy of angle estimation. In fall detection systems, where swift and accurate recognition of changes in body position is critical, the one-dimensional Kalman filter proves beneficial by offering a streamlined yet robust means to continuously monitor and interpret angular changes. Its ability to handle sensor data integration and mitigate measurement errors ensures a more reliable determination of body orientation, aiding in distinguishing normal movements from fall-related dynamics. Consequently, the one-dimensional Kalman filter serves as a valuable tool in fall detection systems, enabling real-time and precise identification of potential falls based on subtle changes in body angle.

- Kalman 1D main fomula:

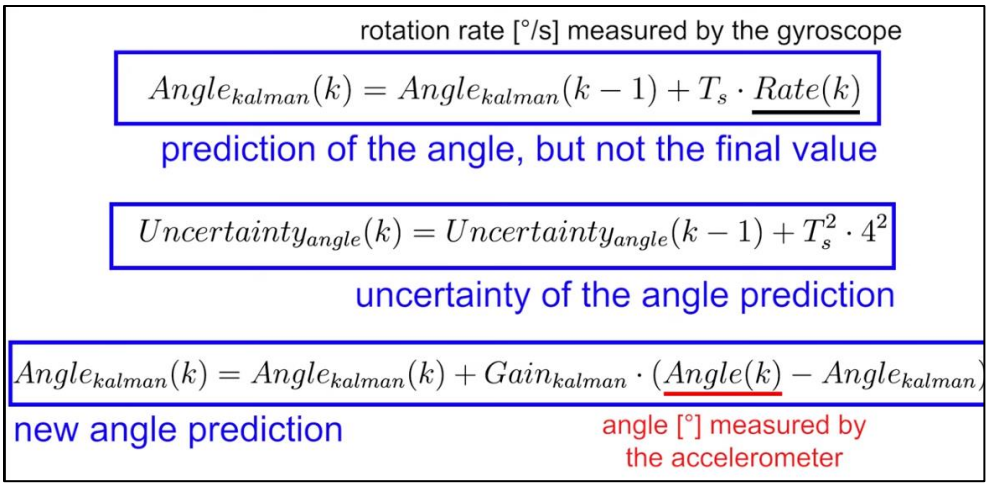


Figure 36. Karman 1-Dimensional Main Fomula

This filter combines the predicted angle with a correction factor based on the difference between the measured angle (calculated using accelerometer trigonometry) and the predicted angle.

The Kalman filter uses a concept known as the Kalman gain, which adjusts the predicted angle by considering both the existing prediction and the difference between the predicted angle and the angle obtained from the accelerometer's measurements. The accelerometer-derived angle represents the 'measured angle' in this context.

By multiplying the Kalman gain with the difference between the predicted angle and the measured angle, the filter updates its prediction. This process leverages the information from both the prediction and the new measurement, adjusting the predicted angle to minimize errors and provide a more accurate estimation that incorporates information from both the prediction and the actual measurement obtained from the accelerometer data.

```
107  float Alpha_Degree()
108  {
109    kalman_1d(KalmanAngleRoll, KalmanUncertaintyAngleRoll, RateRoll, AngleRoll);
110    KalmanAngleRoll=Kalman1DOutput[0];
111    KalmanUncertaintyAngleRoll=Kalman1DOutput[1];
112    return KalmanAngleRoll;
113  }
```

Figure 37. Caculate KalmanAngleRoll

- Kalman Gain fomula:

$$Gain_{kalman} = \frac{Uncertainty_{angle}(k)}{Uncertainty_{angle}(k) + 3^2}$$

std dev angle(k) = 3° (accelerometer)

$$Uncertainty_{angle}(k) = (1 - Gain_{kalman}) \cdot Uncertainty_{angle}(k)$$

uncertainty of new the angle prediction

Figure 38. Kalman Gain Fomula

The Kalman gain serves as a pivotal factor in the Kalman filter, defining the relative ratio between the uncertainty or variance in the predicted angle and the uncertainty in the measured angle. It quantifies how much emphasis or weight the filter places on the new measurement (in this case, obtained from the accelerometer) compared to the existing prediction.

The equation for updating the Kalman gain utilizes the standard deviation of the accelerometer's measurement error. In this scenario, the standard deviation of the accelerometer measurement error, assumed to be 3 degrees, is crucial in determining the weight given to the new measurement. The Kalman gain is derived from a relationship between these uncertainties, adjusting the contribution of the new measurement to the final estimation.

- Breakdown the code for Kalman Filter 1-Dimensional:

```
void kalman_1d(float &KalmanState, float &KalmanUncertainty, float KalmanInput, float KalmanMeasurement) {
    // 1. Prediction phase: Predict the current state of the system
    KalmanState = KalmanState + 0.004 * KalmanInput;
    // 2. Update the uncertainty of the prediction
    KalmanUncertainty = KalmanUncertainty + 0.004 * 0.004 * 4 * 4;
    // 3. Calculate the Kalman gain from the uncertainties on the predictions and measurements
    float measurementError = 3; // Standard deviation of the accelerometer measurement error assumed to be 3 degrees
    float KalmanGain = KalmanUncertainty / (KalmanUncertainty + measurementError * measurementError);
    // 4. Update the predicted state of the system with the measurement of the state through the Kalman gain
    KalmanState = KalmanState + KalmanGain * (KalmanMeasurement - KalmanState);
    // 5. Update the uncertainty of the predicted state
    KalmanUncertainty = (1 - KalmanGain) * KalmanUncertainty;
    // Kalman filter output
    Kalman1DOutput[0] = KalmanState;
    Kalman1DOutput[1] = KalmanUncertainty;
}
```

Figure 39. Kalman 1-Dimensional Filter

1. **Prediction of the current state**:
   $S(k) = F \times S(k-1) + G \times U(k)$
   S = State vector (Angle (Kalman))
   F = State transition matrix (1)
   G = Control matrix (0.004)
   U = Input variable (Rate)
2. **Calculation of the uncertainty of the prediction**:
   $P(k) = F \times P(k-1) \times F^T + Q$
   P = Prediction uncertainty vector (Uncertainty (Angle))
   Q = Process uncertainty ($T_s^2 \times 4^2$)
3. **Calculation of the Kalman gain**:
   $L(k) = H \times P(k) \times H^T + R$
   $K = P(k) \times H^T \times L(k)^{-1}$
   L = Intermediate matrix
   K = Kalman gain
   H = Observation matrix (= 1)
   R = Measurement uncertainty ($T_s^2 \times 3^2$)
4. **Update the predicted state of the system**:
   $S(k) = S(k) + K \times (M(k) - H \times S(k))$
   M = Measurement vector (Angle)
5. **Update the uncertainty of the predicted state**:
   $P(k) = (I - K \times F) \times P(k)$
   I = Unity matrix (= 1)

Figure 40 Walkthrough caculation of KalmanAngleRoll

d) Accquiring thresholds

- Accquisition lower threshold, upper threshold, and orientation fall:

Accelerometer and gyroscope data obtained from an ESP8266 and MPU6050 are leveraged to capture the signals representing the movements. These signals are used to compute the Acceleration-Gravity Vector and Estimated Sensor Resultant (AGVeSR) magnitude. The objective is to discern specific actions, notably falling and jumping, which typically generate substantial acceleration patterns.

This approach facilitates the development of a system that can recognize and differentiate distinct movement patterns, specifically those associated with falls, by examining and evaluating key features extracted from the accelerometer and gyroscope data.

```
//MEASURING THRESHOLD//
  Read_MPU();
  RateRoll -= RateCalibrationRoll;
  RatePitch -= RateCalibrationPitch;
  RateYaw -= RateCalibrationYaw;
  float Amp = sqrt(pow(abs(AccX) + abs(RateRoll), 2) + pow(abs(AccY) + abs(RatePitch), 2)  + pow(abs(AccZ) + abs(RateYaw), 2));
  Serial.print(Amp);
  Serial.print(" ");
  Serial.println(Alpha_Degree());
```

Figure 41. Measuring threshold code

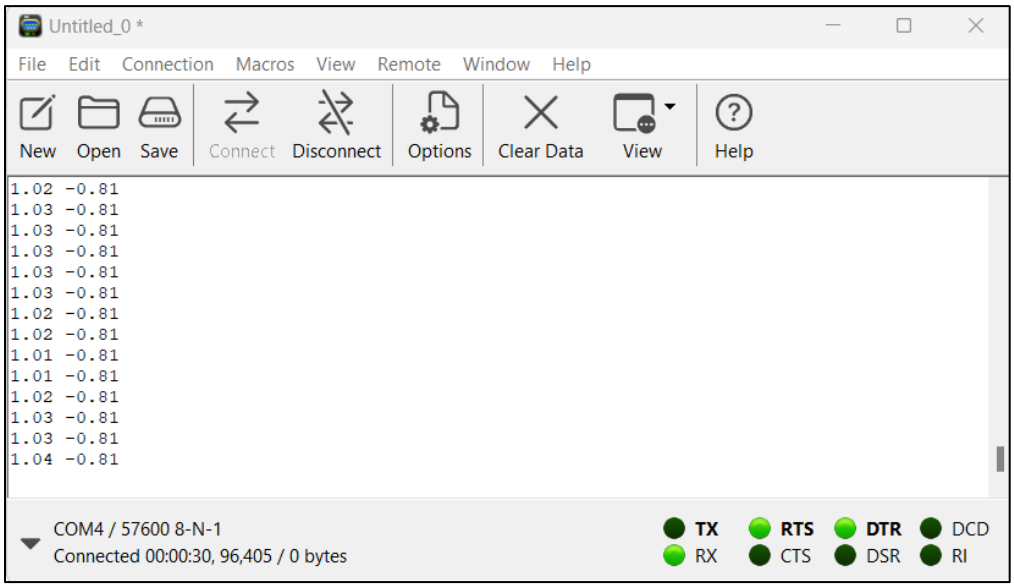


Figure 42. Reading values from COM port using CoolTerm.

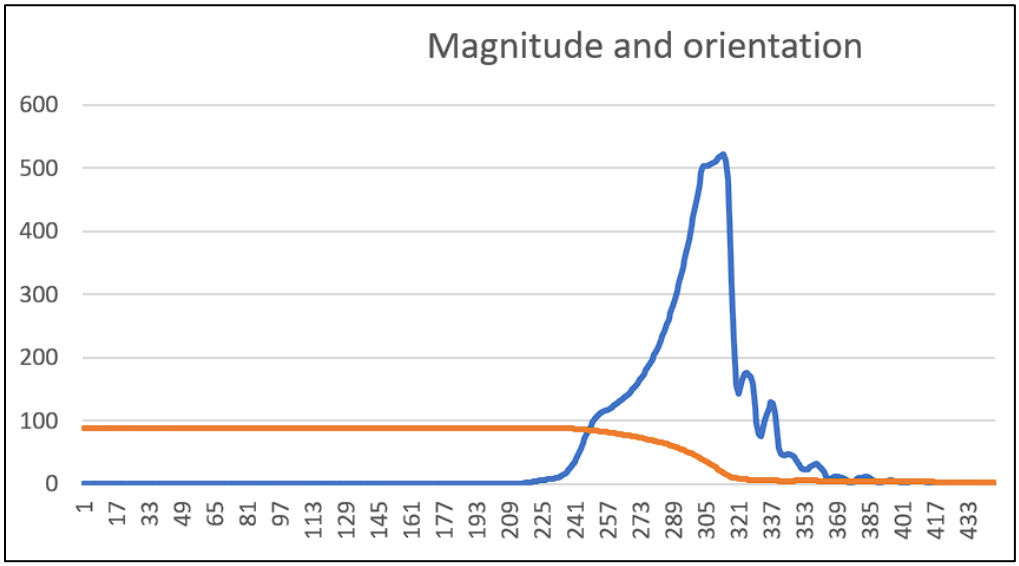


Figure 43. Line chart based on COM port data.

e) AGVeSR method

The AGVeSR (Resultant Acceleration method) introduced in this scenario combines data from both accelerometer and gyroscope sensors to enhance the accuracy of detecting human movements, particularly falls or other daily activities. By integrating information from the six axes—three from the accelerometer (AX, AY, AZ) and three from the gyroscope (GX, GY, GZ)—this method creates a comprehensive view of human motion. The implementation of the AGVeSR formula doesn't rely on specific device orientations but instead uses absolute notation to capture the overall movement irrespective of the sensor's positioning. By considering absolute values, the method normalizes data, ensuring that whether an axis registers a negative or positive value, the resulting absolute value aids in the calculation of sensor data. This approach enhances the sensor's capability to recognize various human activities, regardless of the smartphone's positioning, making it more versatile and adaptable to unconstrained sensor placements.

$$AGVeSR = \sqrt{\left(\left(|AX|+GX\right)^2 + \left(|AY|+GY\right)^2 + \left(|AZ|+GZ\right)^2\right)}$$

Figure 44. AGVeSR fomula

f) The mechanisms of the system.

```
276  if (Amp >= lower_thres && !trigger2 && !trigger3 && !fall) {
277    trigger1 = true;
278    Serial.println("TRIGGER 1 ACTIVATED");
279  }
280
281  if (trigger1) {
282    trigger1count++;
283    if (Amp >= upper_thres) {
284      trigger2 = true;
285      Serial.println("TRIGGER 2 ACTIVATED");
286      trigger1 = false;
287      trigger1count = 0;
288    }
289    // Allow a 0.5-second window for Amp to break the upper threshold for trigger1
290    if (trigger1count >= 125) {
291      trigger1 = false;
292      trigger1count = 0;
293      Serial.println("TRIGGER 1 DEACTIVATED");
294    }
295  }
```

Figure 45. First Trigger as (Impact State)

```
297   if (trigger2) {
298     if (trigger2count < 375) {
299       angle_after = Alpha_Degree();
300       trigger2count++;
301     } else {
302       trigger2count = 0;
303       if (abs(angle_before - angle_after) >= angle_fall) {
304         trigger3 = true;
305         trigger2 = false;
306         Serial.println("TRIGGER 3 ACTIVATED");
307       } else {
308         trigger2 = false;
309         Serial.println("TRIGGER 2 DEACTIVATED");
310       }
311     }
312   }
```

Figure 46. Second Trigger as (Record g after)

```
314   if (trigger3) {
315     if (trigger3count == 0) {
316       delay(8000); // 6-second waiting period
317       trigger3count++;
318     } else {
319       if (count1 < 500) {
320         angle_end = Alpha_Degree();
321         count1++;
322       } else {
323         count1 = 0;
324         if (abs(angle_end - angle_after) <= angle_stable) {
325           Serial.print(angle_end);
326           Serial.print(",");
327           Serial.println(angle_after);
328           fall = true;
329           trigger3 = false;
330           trigger3count = 0;
331           coordinate_noti(); // Send coordinates if a fall might happen
332         } else {
333           Serial.print(angle_end);
334           Serial.print(",");
335           Serial.println(angle_after);
336           trigger3 = false;
337           trigger3count = 0;
338           Serial.println("TRIGGER 3 DEACTIVATED");
339         }
340       }
341     }
342   }
```

Figure 47. Third Trigger as (Record g end)

```
344   // Handle the detected fall
345   if (fall) {
346     fall_noti();
347     Serial.println("FALL DETECTED");
348     Serial.println("Withdraw? PRESS THE BUTTON");
349     if (message) {
350       fall = false;
351       delay(500);
352       fall_noti();
353       message = false;
354     }
355   }
```

Figure 48. Fall detected notification

The program begins by assessing the accelerometer data, initially checking if the values surpass a lower threshold. Upon meeting this condition, the program pauses for half a second before verifying if the values exceed an upper threshold. When the upper threshold is surpassed, the program then proceeds to compute the change in orientation based on the gyroscope data. This change in orientation is monitored for sudden alterations.

Upon detecting a sudden change in orientation, the program initiates a 10-second interval to evaluate if this altered orientation remains consistent over this duration. This sequence of checks and validations allows the program to detect significant changes in movement and orientation. If the orientation remains the same,regard as falling.

3.2.3 Source code

Link google drive:
https://drive.google.com/file/d/1bxaM65nSPOAUfWeA3pnU0fSmNkMlbbMy/view?usp=sharing

CHAPTER 4. EXPERIMENT AND TEST RESULTS

4.1 Threshold establishment

To gauge the accuracy of the proposed models and prototypes, various scenarios were tested. The evaluation process was conducted within two distinct groups, each featuring four different scenarios. The objective was to scrutinize the performance of the proposed approach and its corresponding prototype, which was outfitted with both accelerometer and gyroscope sensors.

4.1.1 First Group

Category	*Scenario*
Walk	Walked
Run	Ran
Squad	Stood up straight -sat down
Sit on chair	Sat on chair

Table I. Activity daily living(ADL) scenarios

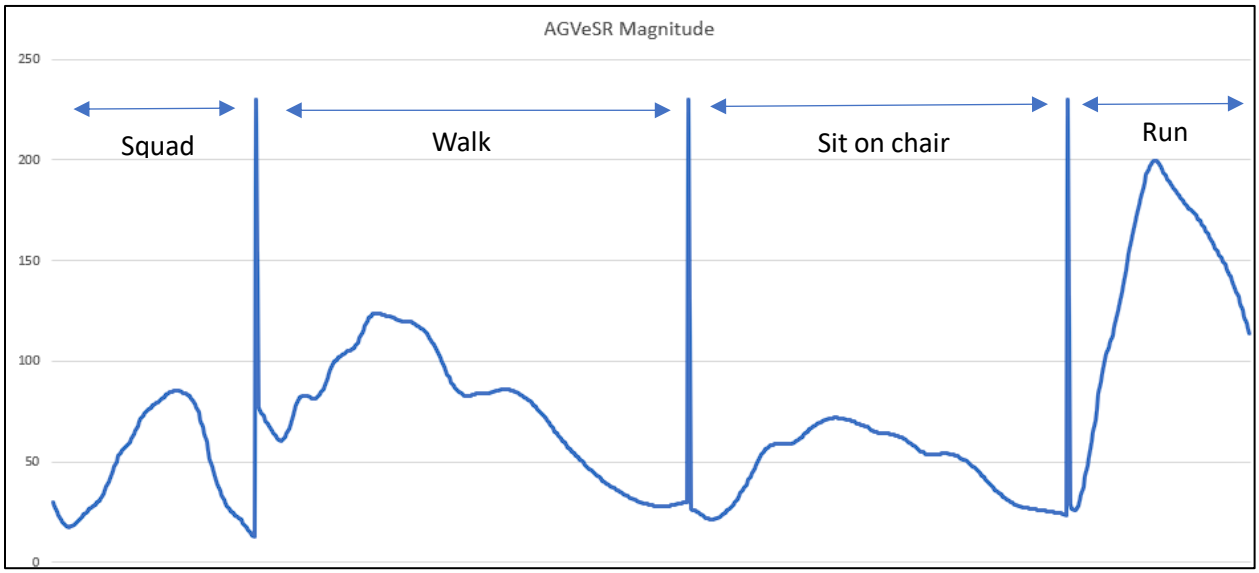


Figure 49. AGVeSR Magnitude - ADL scenarios (Peak Magnitude)

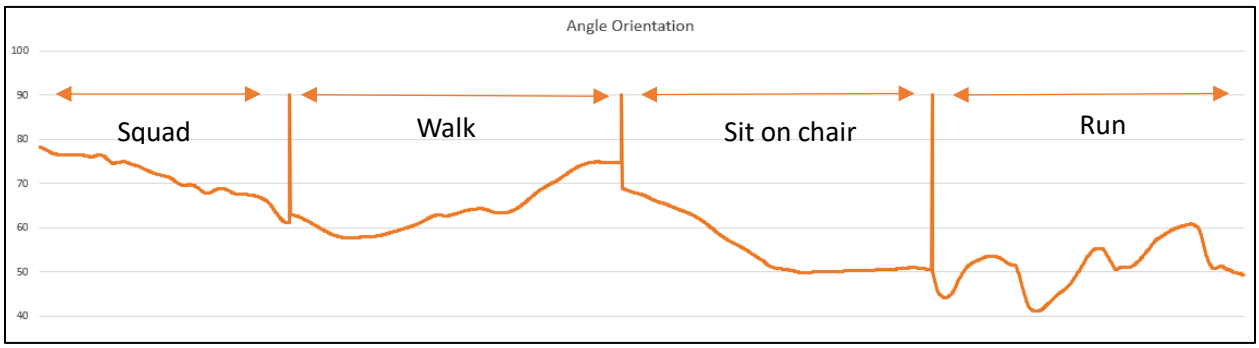


Figure 50. Angle Orientation - ADL scenarios

- *Conclusion*: The AGVeSR magnitude exhibits a peak magnitude of 200, while the associated angle orientation occurs at approximately 20 degrees.

4.1.2 Second Group

Category	*Scenario*
Fall Forward	Walked -fell forward-ended face-down
Fall Backward	Walked -fell backward ended laying on the floor
Fall to the left	Walked -fell to the left ended with laying on the floor
Fall to the right	Walked -fell to the right ended with laying on the floor

Table II. Falling scenario

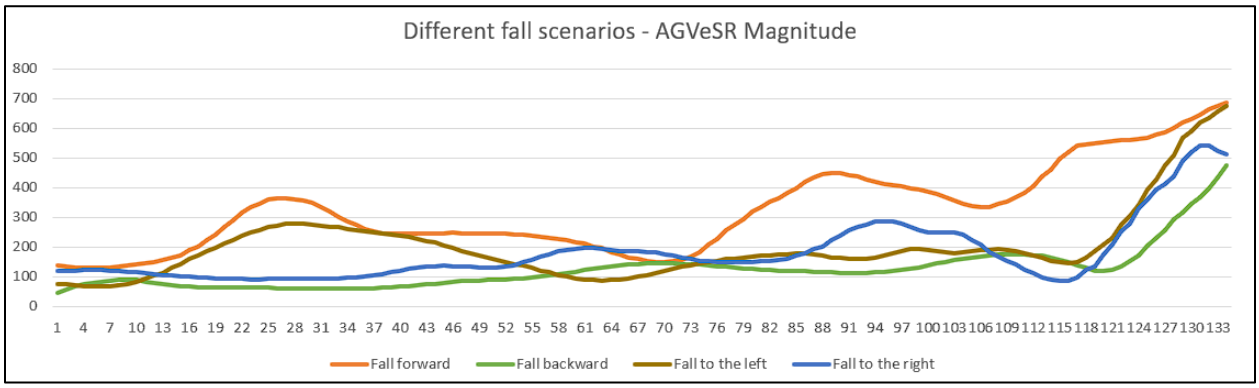


Figure 51. AGVeSR Magnitude - Fall scenarios (Peak magnitude)

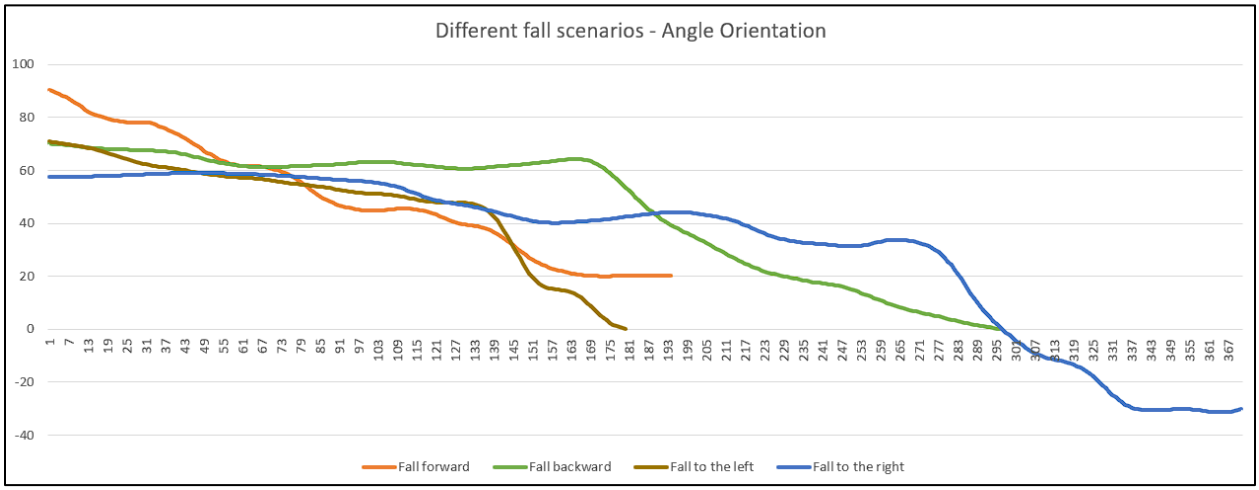


Figure 52. Angle Orientation - Fall scenarios

-*Conclusion*:

+ As illustrated in Figure 49, a discernible trend emerges, indicating that the "fall backward" scenario consistently exhibits the lowest peak magnitude at 491. This value can serve as the upper threshold for all four fall scenarios. Notably, the peak occurs at 133 milliseconds, implying that the lower threshold could occur anywhere from 10 milliseconds onward, given the 0.5-second interval for the subsequent upper threshold break.

+ To establish a specific lower threshold, we can pinpoint at mark 73 milliseconds, corresponding to an average magnitude of all four scenarios are 150. Concurrently, an upper threshold of 450 presents a balanced choice. This refined analysis provides a clearer delineation of the thresholds, enhancing our understanding of the fall scenarios.

+ In Figure 50, the angle orientations for falling to the left and right are approximately 70 and 89, respectively. Fall backward is at 70, and falling forward is around 70 as well. A prudent choice for setting the angle orientation during a fall is 65, ensuring a reliable threshold.

4.2 Experiment setting

Research experiments were conducted utilizing a test subject of 172cm height to investigate fall detection mechanisms. The experimentations employed a defined threshold system outlined in the following algorithm:
Lower Threshold (a): 150
Upper Threshold (a): 450
Angle of Fall (g): 65 degrees
Angle of Stability (g): 10 degrees

- Experimental Setup:
The research was carried out indoors within an apartment environment. The setup included a PCB integrated with an ESP8266 and an MPU6050 sensor. All generated data was seamlessly transmitted to the USB serial port in real-time, enabling immediate access to the serial monitor of Arduino IDE.

- Methodology:
The experiments were systematically executed a total of 160 times. From the conducted experiments, the instances were categorized, distinguishing between the occurrences of falls and the instances related to activities of daily living (ADL).

4.3 Experiment scenario

- The assessment for fall detection was conducted on a padded surface, with the subject of the study being a 55 kg male with a height of 172 cm. The wearable device was positioned within the right lower quadrant of the subject. The following scenario outlines a simulated fall event.

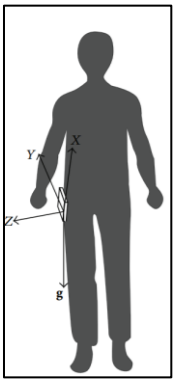


Figure 53. The position of the device

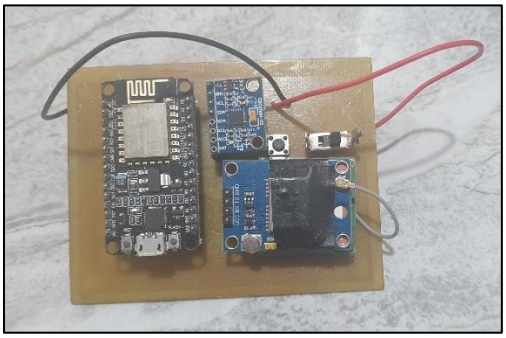


Figure 54. Fall Detection Device

-Several falling scenarios were exhibited by 1 person who experimented on it 20 times in each scenario. ADL Experiments was used to determine the accuracy of fall detection algorithms:

Category	Total	Alarm		Accuracy
		YES	NO	
Fall forward	20	19	1	95%
Fall backward	20	16	4	80%
Fall to the left	20	19	1	95%
Fall to the right	20	18	2	90%

Table III. Falling result test

Table III showed the accuracy of fall detection algorithm that was applied. Each scenario were carried 20 times for maximum clearance.

Category	Total	Alarm		Accuracy
		YES	NO	
Walk	20	0	20	100%
Run	20	2	18	90%
Squad	20	0	20	100%
Sit on chair	20	2	18	90%

Table IV. Activity daily living result (ADL)

Based on Table IV, the algorithm still detected the occurrence of falling in some daily activities. For instance, during the ADL experiment, 2 of 20 attempts at running were detected as a falling state. When sitting down there were 2 of 20 attempts that were detected as a falling state. In another case, moving down the stairs was also detected as a falling state due to the gravity level affecting the acceleration of the detector device.

CHAPTER 5. SUMMARY, RESTRICTIONS AND DEVELOPMENT ORIENTATIONS

5.1 Summary

The development and assessment of a fall detection system tailored for assisting the elderly have yielded promising results. The prototype, strategically positioned at the right lower quadrant of the subject, proficiently gathered sensor data and alongside a threshold detection algorithm for accurate fall detection.

In a series of experiments consisting of 60 simulated falls, the system exhibited an impressive accuracy rate of 90%. Additionally, across 60 comprehensive experimental trials focusing on activities of daily living (ADL), the system showcased an average accuracy of 95%.

5.2 Limitations

Component Arrangement: The existing setup lacks a consolidated and portable casing, presenting challenges in terms of portability and user-friendliness. The current configuration, mounted on a PCB, hinders convenient mobility.

GPS Functionality: The detachment of the NEO-6M GPS indoors underscores the limitation of indoor functionality. There is a need for further exploration to improve its reliability in indoor settings or explore alternative location detection mechanisms.

5.3 Future Development Directions

The system's capability to swiftly detect a fall event and initiate an alarm represents a significant advancement in ensuring prompt response and intervention. However, it is crucial to recognize that the system's capabilities may need refinement to effectively handle various falling scenarios that could occur in real-world settings.

While this research predominantly emphasizes detection mechanisms, future endeavors could delve into additional features. Integrating functionalities such as the ability to send short messages (SMS) and obtaining precise positioning of the wearable device would substantially enhance the system's efficacy, facilitating timely assistance during emergencies.

REFERENCES

[1]: Research Article - Development of a Wearable-Sensor-Based Fall Detection System
School of Instrumentation Science and Optoelectronics Engineering, Beihang University, Beijing 100191, China2 Space Star Technology Co., Ltd., Beijing 100086, China
[2]: Accurate, Fast Fall Detection Using Gyroscopes and Accelerometer-Derived Posture Information – Qiang Li, John A. Stankovic, Mark Hanson, Adam Barth, John Lach University of Virginia
[3]: Human Fall Detection using Accelerometer and Gyroscope Sensors in Unconstrained Smartphone Positions - Maria Seraphina Astriani, Yaya Heryadi, Gede Putra Kusuma, Edi Abdurachman
[4]: Fall Detection System Using Accelerometer and Gyroscope Based on Smartphone - Arkham Zahri Rakhmani , Lukito Edi Nugrohoi , Widyawani , Kurnianingsihi , 2 IDept. of Electrical Engineering and Information Technolog
[5]: Combine a gyroscope and accelerometer to measure angles – precisely - Carbon Aeronautics

LIST OF FIGURES

LIST OF TABLES

CONTRIBUTION

Task Assignment Table	
Members	Tasks
Trương Hữu Trường Sơn – 21522559	-Write report -Design Fall Detection Algorithm -Develop App Interface -Manage Sensor Data Accquisition
Đỗ Thanh Sơn – 21522551	-Conducting Experiments -PCB Manufacture -Minor contributions to App Interface