**School of Professional Education and Executive Development**

**The Hong Kong Polytechnic University**

**Integrated Study submission for the award of the**

**Bachelor of Science (Honours) Scheme in Applied Sciences**

**FALL DETECTION USING IOT DEVICE FOR ELDERLY**

**Final Report**

**THAPA SANJAY**

**21070979S**

**Supervisor: Dr James Chau**

**Number of words: 3,776**

**Submission date: 24-Apr-2023**

**INTEGRATED STUDY REPORT DECLARATION FORM**

I, THAPA SANJAY (student no.: 21070979S), hereby declare the Integrated Study report, the title of which is FALL DETECTION USING IOT DEVICE FOR ELDERLY, submitted on 28/04/2023 for PolyU SPEED programme Bachelor of Science (Honours) Scheme in Applied Sciences, is my work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor the work of other students, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

Word Count: 3,776 words

Text

Description automatically generatedSigned by:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
 (Signature of student)

Date: 28/04/2023

**Table of Contents**

[Abstract 4](#_Toc133593005)

[Introduction 4](#_Toc133593006)

[Industry and Market Analysis 4](#_Toc133593007)

[Software Requirements 7](#_Toc133593008)

[Algorithm Design 7](#_Toc133593009)

[System Design 10](#_Toc133593010)

[System Implementation 12](#_Toc133593011)

[Algorithm Implementation 12](#_Toc133593012)

[Fall Detection Algorithm Implementation 12](#_Toc133593013)

[Amazon Web Service Connection 16](#_Toc133593014)

[Executing the code 17](#_Toc133593015)

[Cloud Implementation 19](#_Toc133593016)

[Testing Approach 19](#_Toc133593017)

[Testing Results and Evaluation 20](#_Toc133593018)

[Noise Detection Test 20](#_Toc133593019)

[Fall Detection Test 20](#_Toc133593020)

[Evaluation 21](#_Toc133593021)

[Demonstration 22](#_Toc133593022)

[Conclusion 23](#_Toc133593023)

[References 24](#_Toc133593024)

[Appendices 26](#_Toc133593025)

[C++ code to Test connection to AWS IoT Service 26](#_Toc133593026)

[C++ Final Project Code 29](#_Toc133593027)

# Abstract

This research focuses on developing a fall detection system for the elderly using Internet of Things (IoT) devices. The study includes an analysis of the industry's current state regarding IoT devices for the elderly, along with their advantages and disadvantages. Additionally, the research examines the software used to develop the fall detection system and how it utilizes IoT hardware to gather data. The testing section describes the sampling technique used to conduct the tests. Finally, the study evaluates the results and draws conclusions about the effectiveness of the system. Overall, this research aims to fill the gap in the market and provide a solution to the rising concern of falls among the elderly.

# Introduction

In Hong Kong, an elderly man was discovered unconscious in an inaccessible restroom adjacent to the women’s restroom at the Hung Hom MTR station. He was pronounced dead later that night at the Queen Elizabet’s hospital (Chau, 2022). It is highly likely the premortem to be a fall, which rendered him unconscious. This unfortunate event could have been prevented if a message of his location had been transmitted to the family. Researched published in the Hong Kong Medical Journal indicated that “fallers commonly experience a decline in physical functions and the activity of daily living” (Chu et al., 2007), highlighting the significance of promptly detecting falls and transmitting information to relevant parties. Preventable deaths and permanent physical function declines could be avoided with such technology. Consequently, the objective of this study is to develop and IoT device that can detect falls among the elderly in near real time.

This study provides a significant contribution to the knowledge and application of IoT devices in addressing societal issues. No isolation states that “learning new technology can be challenging for seniors who lack experience in using technology as a baseline. Seniors have limited reference points to absorb new knowledge” (No Isolation, n.d.). Part of out focus will be on creating a suitable user experience for the elderly, as have difficulties adapting to modern technology. Furthermore, this study also contributes to the development of smart IoT devices, the information gathered in this research can be adapted to other areas, such as detecting fatal illness in elderlies.

# Industry and Market Analysis

The elderly care market is very diverse as it contains a lot of other markets, such as smart home devices & wearables. All these markets are mutually inclusive as they share a common goal of improving the quality of life of the elderly. As mentioned by Cogniteq (2022), the global IoT market volume hit $384.70 billion in 2021, and has a projected Compound Annual Growth Rate (CAGR) of 26.4% for the period from 2022 to 2029. The article mentions the COVID-19 pandemic is a factor that increased the adoption of smart devices among elderly. This implies that elderly and caregiver are adapting new devices to accommodate to their care needs. The article reference Pew Research Centre, which 92% of the people in the US aged 65 and older own a mobile phone. With these figures, the market of elderly care solutions has a promising growth.

The market can be segmented into three areas security & safety, Health State monitoring, and Everyday tasks. Security and safety for seniors, many seniors love alone without any care givers, and the elderly are vulnerable to accidents & emergencies. Smart Home devices can help mitigate these risks by providing real-time alerts on falls or other emergency situations. Health State monitoring, health is the most important factor of elderly, as majority of them develop chronic disease which requires constant monitoring. For example, glucometer measures how much sugar is in the blood sample (University of Rochester). Everyday tasks, these devices assist elderly perform many tasks. For example, elderly have difficulty hearing, earphones to amplify the voices can help the elderly hear. The researcher of this study believes the IoT device built solely focuses on Security and safety for seniors and may overlap other areas.

The three largest companies in the elderly care market are Koninklijke Philips N.V., Unicharm, and Cardinal Health. This analysis focuses on the Philips Lifeline Home Safe Standard Support, which costs USD29.95 for the basic setup and USD50 for activation. The setup includes a base communicator unit that requires a landline and an alarm button in either a pendant or wristband style. Both styles are comfortable, but the device requires a landline, which can increase setup costs, and inconsistent customer service experiences have been reported. An alternative option is the On the Go Mobile Device, which costs USD49.95 and has an activation fee of USD99.95. In the event of a fall, the alarm button triggers a signal to the communicator, which alerts the Lifeline 24/7 response center. For falls outside the home, the On-the-Go Mobile Device pendant will call the response center and send the location.

|  |  |  |  |
| --- | --- | --- | --- |
| Product | Monthly Fee | Activation Fee | Sum |
| Home safe Standard Support | USD 29.95  HKD 235.11 | USD 50  HKD 392.50 | USD 79.95  HKD 627.60 |
| On the Go Mobile device | USD 49.95  HKD 392.11 | USD 99.95  HKD 784.60 | USD 149.9  HKD 1,176.71 |

HKD Price on 04/2023

The IoT device used in this study is like the On-the-Go lifeline device, as both are wearable devices that can be taken anywhere. However, upon examining the device's specifications, several technological gaps become apparent, such as the lack of integration with devices other than their own, the high monthly and activation fees, and the vendor lock-in. The device only supports its own emergency response system. The objective of this study's IoT device is to address these technological gaps by making it affordable, easily integrated with the owner's devices, and vendor neutral.

**Technical Knowledge Requirements**

To conduct this study and implement the fall detection algorithm, a strong grasp of mathematics is necessary. Specifically, knowledge of acceleration and angular velocity is required to calculate data from the gyroscope Additionally, other mathematical techniques such as standard deviation are needed to smooth out the accelerometer data and gyroscope data. Readers & developers must have a good understanding of these concepts to understand the logic of the fall detection algorithm. However, with this understanding, developers can easily utilize the available math libraries to implement the algorithm.

|  |  |
| --- | --- |
| Concept | Equation |
| Acceleration Vector | *Text  Description automatically generated* |
| Angular Velocity | A picture containing text  Description automatically generated |
| Standard Deviation | Text  Description automatically generated with medium confidence |
| Order of magnitude | A picture containing shape  Description automatically generated |

Apart from the mathematics and software requirements, a general understanding of electrical engineering is also required to understand the IoT device. This study uses Signal Processing techniques such as filtering and smoothing. Filtering is the process of removing unwanted noise from a sensor. Smoothing, is the process of reducing variability from a sensor. It can make it easier to detect patterns of falls. Both techniques help reduce the impact of noise and variability, making it easier to detect falls. However too much smoothing and filtering can cause lower accuracy.

# Software Requirements

The above-mentioned products are electronic device, they mostly contain firmware and protocols. For our development of an IoT device, we will be using C++ as the programming language, C++ is an object-oriented programming language popular for embedded device, the C++ language will help us communicate with the microcontroller to provide instruction on the fall detection algorithm. To control and interact with the IoT devices, we have chosen to use the Arduino Integrated Development Environment (IDE) within our study. The Arduino IDE provides us the code editor, board manager, serial monitor, library manager and debugger. All necessary for the algorithm development. Details of the M5StickC Firmware is not provided.

In the M5StickC documentation, it lists several methods to program the IoT device, such as Python, block-based programming, C#. The Researcher has chosen C++ and Arduino IDE due to the high level of experience in this software.

In addition to the necessary software for development, the research also used a project management web application called Online Gantt. It helps the researcher plan, schedule, and track progress of the project against the ideal timeline and helps identify potential delays or issues. The study used this divide the goals of research & preparation, software development & testing, and writing the report.

|  |  |
| --- | --- |
| Software | Version |
| C++ | C++ 11 Arduino variant |
| Arduino IDE | 2.0.5 |

This study opted to use Amazon Web Services (AWS) and MQTT for the communication between the IoT device and the cloud. Specifically, the M5Stack device was chosen due to its extensive documentation and the availability of an AWS IoT Software Development Kit (SDK) for C++. In order to work with AWS IoT, it is recommended to have a good understanding of MQTT and its messaging protocol. While HTTP requests and REST APIs are important for cloud development, AWS IoT offers a low-code platform that requires minimal configuration to establish communication with the wearable device.

# Algorithm Design

The initial stage of the design was very straightforward in calculating the acceleration and angular velocity of the device, if both measurement passes the threshold, then the device returns true for fall detected.

Diagram

Description automatically generated

However, the accuracy was poor on the first trial, sometimes the fall detection triggered randomly. The sensor data was messy and required smoothing. Sensor smoothing was employed as a technique to adapt to long-term changes rather than being triggered by short-term variations (Sumida Crossing, n.d.). Several smoothing algorithms were available, but for this case, a simple smoothing algorithm was selected. There is various smoothing algorithm available, for this case a simple smoothing algorithm.

The smoothing algorithms has a window size of 50, indicating the number of adjacent data points to be averaged. Loops through the input data vector starting at the first index and stopping at the last index minus the window size. At each iteration, it calculated the sum of the data points within the window, computed the average by dividing the sum by the window size. Finally return the smoothed data vector.

The new Algorithm Logic with the added smoothing algorithm.

Diagram

Description automatically generated

The major difference in the second version is the comparison of the standard deviation for the smoothed accelerometer data to the threshold. The threshold value is a predetermined value that is set based on the characteristics of the individual using the fall detection system. It represents the maximum amount of variability in the smoothed accelerometer data that can be tolerated before a fall is detected. If the standard deviation of the smoothed accelerometer data exceeds the threshold value, the program considers that a fall has occurred and returns a "true" value. Conversely, if the standard deviation of the smoothed accelerometer data is below the threshold value, the program considers that no fall has occurred and returns a "false" value.

# System Design

First and foremost, we are using the M5StickC as the wearable device, which will communicate to the cloud and other devices. M5StickC contains a lot of specification, this section will only mention the relevant ones to our discussion. It contains an ESP32-based microcontroller, with 240MHz dual core & 520KB SRAM, Wi-Fi enabled, Type C port, 0.96 LCD Screen, 6-Axis Inertial Measurement Unit. The researcher would like to include Power Management Unit as a factor in system design, since with the limited computing resource, the code had to be optimized to take minimum memory. In addition, A 3-meter cable is required to recharge the portable IoT device, providing additional support to its battery and enabling it to be charged from anywhere.

To transmit data between the IoT device and caretakers, a cloud provider is necessary. For this study, Amazon Web Services (AWS) was utilized, which provides Internet of Things services to connect the device to the cloud and receive and process telemetry data. Along with the basic features, AWS offers additional features such as device management, data management, and integration. To process the data, AWS Lambda will be used, while Amazon DynamoDB will be used for storage.

AWS Lambda is a serverless computing service that enables developers to execute code without the need to manage or provision servers. In this project, AWS Lambda will be used to handle incoming telemetry data and process it before sending it to the appropriate third party. The data will be sent using the Message Queuing Telemetry Transport (MQTT) protocol, which can be easily converted to JSON documents by developers. This makes DynamoDB a suitable choice for storing the data, as it provides low latency and scalable performance, with automatic scaling and partitioning of data across multiple servers to handle growing workloads. Additionally, DynamoDB can be easily integrated with other services that will be used in this project, such as the AWS Lambda function.

Lastly a crucial part of our system design is computer networking protocols, specifically Transport Layer protocols, AWS has strict security protocols to be able transmit data to the AWS endpoint, this project uses a cipher suite, which are Elliptic Curve Diffie-Hellman Ephemeral (ECDHE), Rivest-Shamir-Adleman (RSA), Advanced Encryption Standard with a 128-bit key (AES-128), Galois/Counter Mode (GCM), and Secure Hash Algorithm 256-bit (SHA256). With a mix of encryption protocol, it is highly unlikely malicious hacker can interfere with the IoT devices and their transmission. Common knowledge of Wi-Fi connection and router setup is also required

Graphical user interface, application

Description automatically generated

*Figure 1: High level architecture diagram*

The high-level architecture diagram will illustrate the process of detecting falls in elderly people using Internet of Things. When a fall is detected, the IoT device will publish a message to the AWS Cloud using the MQTT protocol, which includes the phone number and a message. The data will be processed by an AWS Lambda function, which will then store it in JSON format in DynamoDB and send a message to the caretakers. It is important to note that the transmission of messages requires an internet connection, and if the connection is not available, the fall detection message will not be received by the AWS Cloud.

Diagram

Description automatically generated*Figure 2: Architecture Diagram of AWS cloud*

Within the AWS Cloud, numerous services work together to process data and convert it into actionable information for this project's continued improvement. Initially, the IoT device sends a message using the MQTT protocol to the AWS IoT Core service, which then delivers the message to the designated subscriptions. Subsequently, a Lambda function processes the data and publishes a message to the designated topic. Finally, this message is sent to the caretaker's device using the MQTT protocol.

# System Implementation

All the software and system design were successfully implemented as planned. However, a different method of transmission protocol was made due to the requirements of the cloud provider.

## Algorithm Implementation

### Fall Detection Algorithm Implementation

The threshold for detection fall would be 0.3 ms−2. The Window size for smoothing the data is 50.

//Define constants

const double THRESHOLD = 0.3;  //Fall detection threshold in m/s^2

const int WINDOW\_SIZE = 50;    //window size for smoothing data

To store the data structure in appropriate format, the code defines 2 data structures, “**AccelerometerData**” and “**GyroscopeData**”. Each structure contains three float values for the x, y, and z component of the accelerometer and gyroscope data, respectively.

//Define data structures

struct AccelerometerData {

  float x, y, z;

};

struct GyroscopeData {

  float x, y, z;

};

This function calculates the magnitude of the acceleration vector using the Euclidean distance formula. It takes the **AccelerometerData data structure** as the input and returns the magnitude of the vector.

//Calculate magnitude of acceleration vector

float calculateMagnitude(AccelerometerData data) {

  return sqrt(pow(data.x, 2) + pow(data.y, 2) + pow(data.z, 2));

}

This function smooths accelerometer data using a sliding window algorithm. It calculates the average value of each window of size "WINDOW\_SIZE" and adds it to a new vector called "smoothedData". The function then returns the smoothed data as a vector of floating-point values. This helps to remove high-frequency noise or fluctuations in the raw data and obtain a more accurate representation of the signal.This code detects falls based on accelerometer and gyroscope data. It calculates the standard deviation of the z-axis accelerometer data using a sliding window of "WINDOW\_SIZE" data points. If the standard deviation is above a certain threshold defined as "THRESHOLD", the function returns "true" to indicate that a fall has been detected, otherwise it returns "false".

//Smooth Accelerometer data using a sliding window

std::vector<float> smoothData(std::vector<float> data) {

  std::vector<float> smoothedData;

  for (int i = 0; i <= data.size() - WINDOW\_SIZE; i++) {

    float sum = 0;

    for (int j = i; j < i + WINDOW\_SIZE; j++) {

      sum += data[j];

    }

    smoothedData.push\_back(sum / WINDOW\_SIZE);

  }

  return smoothedData;

}

// Function to detect falls based on accelerometer and gyroscope data

bool detectFall(const std::vector<AccelerometerData>& accelerometerData, const std::vector<GyroscopeData>& gyroscopeData) {

  // Calculate standard deviation of the z-axis accelerometer data

  float mean = 0.0;

  for (int i = 0; i < WINDOW\_SIZE; i++) {

    mean += accelerometerData[i].z;

  }

  mean /= WINDOW\_SIZE;

  float stdDev = 0.0;

  for (int i = 0; i < WINDOW\_SIZE; i++) {

    stdDev += pow(accelerometerData[i].z - mean, 2);

  }

  stdDev = sqrt(stdDev / WINDOW\_SIZE);

  // Print standard deviation to serial monitor

  Serial.print("Standard deviation: ");

  Serial.println(stdDev);

  // Detect fall if the standard deviation is above the threshold

  return stdDev > THRESHOLD;

}

This is a code snippet that sets the cursor position on a graphical display to improve text visibility. The function takes an integer input called "lineNumber" which specifies the line number on the display where the cursor should be set. It then uses predefined constant values for the x and y position of the cursor and calculates the vertical position based on the line number and line height. The function then sets the text size of the cursor to 2x the default size. This code can be used with other display functions to draw text or graphics at a specific location on the display.

void setCursorToLine(int lineNumber) {

  const int X\_LOCATION = 5;

  const int Y\_LOCATION = 0;

  const int LINE\_HEIGHT = 10;

  M5.Lcd.setCursor(X\_LOCATION, Y\_LOCATION + ((lineNumber - 1) \* LINE\_HEIGHT), 2);

}

This is a code used for printing accelerometer and gyroscope data to the serial monitor. It loops through each element in the "accelerometerData" vector, and for each element, it prints the x, y, and z-axis values to the serial monitor.

void printData(const std::vector<AccelerometerData>& accelerometerData,

const std::vector<GyroscopeData>& gyroscopeData)

{

  Serial.println("Accelerometer data:");

  for (const auto& data : accelerometerData) {

    Serial.printf("X=%.2f, Y=%.2f, Z=%.2f\n", data.x, data.y, data.z);

  }

  Serial.println("Gyroscope data:");

  for (const auto& data : gyroscopeData) {

    Serial.printf("X=%.2f, Y=%.2f, Z=%.2f\n", data.x, data.y, data.z);

  }

}

### Amazon Web Service Connection

This code connects an ESP32 device to the AWS IoT Core service using the MQTT protocol. It connects to a WiFi network using the SSID and password, and then configures a WiFiClientSecure object to use the AWS IoT device credentials. The function creates a message handler and subscribes to an MQTT topic. If the connection is successful, the function prints a message to the serial monitor and the display.

void connectAWS()

{

  WiFi.mode(WIFI\_STA);

  WiFi.begin(WIFI\_SSID, WIFI\_PASSWORD);

  Serial.println("Connecting to Wi-Fi");

  while (WiFi.status() != WL\_CONNECTED){

    delay(500);

    Serial.print(".");

  }

  // Configure WiFiClientSecure to use the AWS IoT device credentials

  net.setCACert(AWS\_CERT\_CA);

  net.setCertificate(AWS\_CERT\_CRT);

  net.setPrivateKey(AWS\_CERT\_PRIVATE);

  // Connect to the MQTT broker on the AWS endpoint we defined earlier

  client.begin(AWS\_IOT\_ENDPOINT, 8883, net);

  // Create a message handler

  client.onMessage(messageHandler);

  Serial.print("Connecting to AWS IOT");

  if (!client.connect("MyNewESP32")) {

    Serial.print(".");

    delay(100);

  }

  if(!client.connected()){

    Serial.println("AWS IoT Timeout!");

    return;

  }

  // Subscribe to a topic

  client.subscribe(AWS\_IOT\_SUBSCRIBE\_TOPIC);

  Serial.println("AWS IoT Connected!");

  M5.Lcd.println("AWS IoT Connected!");

}

This code sends a message to AWS IoT Core service using the MQTT protocol. The message includes a phone number and a help message and is sent to an MQTT topic using the AWS\_IOT\_PUBLISH\_TOPIC constant defined in the code.

void publishMessage()

{

  StaticJsonDocument<200> doc;

  doc["sms"] = "+14804224242";

  doc["message"] = "Help! I've fallen and I can't get up!";

  char jsonBuffer[512];

  serializeJson(doc, jsonBuffer); // print to client

  client.publish(AWS\_IOT\_PUBLISH\_TOPIC, jsonBuffer);

}

### Executing the code

This code initializes the M5StickC object, clears the display screen, starts serial communication, initializes the IMU sensor, and establishes a secure MQTT connection with the AWS IoT Core service using the "connectAWS" function. This sets up the necessary components for the M5StickC device to communicate with the AWS IoT Core service and initializes the display screen and IMU sensor for the application.

void setup() {

  // Initialize the M5StickC object

  M5.begin();

  M5.Lcd.fillScreen(BLACK);

  Serial.begin(115200);

  M5.IMU.Init();

  connectAWS();

}

This code repeatedly collects accelerometer and gyroscope data for 200 iterations, resizes the data vectors, calls the "detectFall" function to determine if a fall has occurred, and sends an alert message to an MQTT topic using the "publishMessage" function if a fall is detected. It also displays the collected data on the M5StickC display and serial monitor. The loop repeats with a delay of 1 second before collecting more data again.

//Run repeatedly

void loop() {

  // Collect accelerometer and gyroscope data

  std::vector<AccelerometerData> accelerometerData;

  std::vector<GyroscopeData> gyroscopeData;

  for (int i = 0; i < 200; i++) {

    AccelerometerData accelData;

    GyroscopeData gyroData;

    M5.IMU.getAccelData((float\*)&accelData.x, (float\*)&accelData.y, (float\*)&accelData.z);

    M5.IMU.getGyroData((float\*)&gyroData.x, (float\*)&gyroData.y, (float\*)&gyroData.z);

    accelerometerData.push\_back(accelData);

    gyroscopeData.push\_back(gyroData);

    delay(10);

  }

  // Resize the data vectors to the actual number of data points collected

  accelerometerData.resize(200);

  gyroscopeData.resize(200);

  printData(accelerometerData, gyroscopeData);

  //Detect falls

  bool fallDetected = detectFall(accelerometerData, gyroscopeData);

  // Print result

  if (fallDetected) {

    setCursorToLine(3);

    M5.Lcd.printf("Fall detected!");

    Serial.println("Fall detected!");

    publishMessage();

  } else {

    setCursorToLine(3);

    M5.Lcd.printf("No fall detected.");

    Serial.println("No fall detected.");

  }

  // Wait before collecting more data

  delay(1000);

}

## Cloud Implementation

From the M5Stack documentation, it mentions the project only requires a setup of the Aws IoT service. The step only required obtaining the data to implement in the code.

During the setup process, the data such as the AWS IoT Endpoint, public key files, private key file, and Device certificate.

Setup of the topic of the subscription is needed to receive the messages from the IoT device

# Testing Approach

To test the projects, fall detection algorithms, a convenience sampling is used. For testing, 5 participants have volunteered from the Polytechnic University, Hung Hom Bay Campus. For confidentiality reasons, names, and information irrelevant to the project will not be given from the participants.

The gender of the participants are 4 men, and 1 woman. The age ranges are 22-28. The height of chest ranges from 142.6 to 158.4cm for men. The women chest height is 138.3cm. All of them are physically healthy.

In the Noise Detection test, participants will be asked to perform a series of daily activities for 5 minutes, such as walking, talking, and using appliances. The activities will be repeated multiple times to ensure consistency and accuracy of the results. The performance of the noise detection system will be evaluated based on the false fall detected.

In the Fall Detection test, the IoT device will be subjected to a drop test, where the device will be dropped from a predetermined height several times. Additionally, fake attempts to drop the device will be made randomly to test the false positive rate of the fall detection algorithm. The number of drops and fake attempts will be recorded, and the performance of the algorithm will be evaluated based on the number of true positives, false positives, true negatives, and false negatives generated by the algorithm.

# Testing Results and Evaluation

## Noise Detection Test

In the test, the IoT device will be strapped around the waist levels and do general activities for 5 minutes, this will identify the effectiveness of the smoothing algorithm implemented in the code.

|  |  |
| --- | --- |
| Participant | Fall detected |
| Participant 1 | 0 |
| Participant 2 | 0 |
| Participant 3 | 0 |
| Participant 4 | 0 |
| Participant 5 | 0 |

As can be seen in the table, 0 falls were detected when the participants were performing general activity. This proves the effectiveness of the smoothing algorithm implemented in the code to prevent noise.

## Fall Detection Test

To assess the accuracy and precision of fall detection, an additional test involves simulating falls by dropping the device multiple times. This test is conducted five times.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Participant | Attempts | True Drops | False Drops | Fall detected |
| Participant 1 | 10 | 5 | 5 | 3 |
| Participant 2 | 8 | 3 | 4 | 3 |
| Participant 3 | 12 | 6 | 6 | 4 |
| Participant 4 | 14 | 6 | 8 | 5 |
| Participant 5 | 12 | 6 | 6 | 4 |

## Evaluation

Assuming that the "Fall detected" column indicates the predicted class, and the "True Drops" and "False Drops" columns indicate the actual class.

A picture containing graphical user interface

Description automatically generated

Graphical user interface

Description automatically generated

The fall detection device has an accuracy of 77.78%, which is close to the standard for such devices. It can classify false falls as false with a specificity of 57.14%, and it can accurately identify falls as falls with a precision of 68.42%. The F1 score of 81.25% indicates a good balance between precision and recall, demonstrating that the device is both accurate and comprehensive in detecting falls.

The sensitivity is 1, a high sensitivity is especially important because falls can be life-threatening and early detection can prevent serious injury or even death.

# Demonstration

A picture containing text, person, indoor, orange

Description automatically generated A picture containing text, person, indoor, hand

Description automatically generated

*Figure 4: Fall Detected Message Figure 5: No Fall Detected*

Graphical user interface, text, application

Description automatically generated

*Figure 6: AWS Message from fall detection*

As mentioned above, after the fall is detected the IoT device will send a message to AWS cloud with an emergency message.

Link to the video demonstration:

# Conclusion

In conclusion, this project demonstrates the implementation of an IoT-based fall detection system using an ESP32 device and AWS IoT Core service. The code establishes a secure MQTT connection to the AWS IoT service and sends alert messages to a subscribed topic when a fall is detected. The fall detection algorithm was tested using a drop test, simulating falls, and the results showed an accuracy of 77.78%, specificity of 57.14%, precision of 68.42%, and sensitivity of 1. The system also underwent a noise detection test, which proved the effectiveness of the smoothing algorithm implemented in the code.

Overall, this project provides a proof of concept for a fall detection system that can potentially save lives and improve the quality of life for elderly individuals. Further improvements can be made in the algorithm and testing methodology to enhance the accuracy and reliability of the system. For instance, machine learning algorithms such as decision trees, support vector machines, and neural networks can be trained on a larger dataset to identify patterns and improve the accuracy of fall detection.

# References

Connecting AWS on Arduino. m5Stack. (n.d.). <https://docs.m5stack.com/en/arduino/iotcloud/aws>

Forbes Magazine. (2023, April 5). Philips Lifeline Review. *Forbes*. <https://www.forbes.com/health/healthy-aging/philips-lifeline-review/>

Glucose Monitoring Devices. Health Encyclopedia*. University of Rochester Medical Center.* (n.d.). <https://www.urmc.rochester.edu/encyclopedia/content.aspx?contenttypeid=85&contentid=P00339#:~:text=A%20small%20device%20called%20a,that%20quickly%20pricks%20the%20fingertip>.

Global elderly care products and Services Market Report 2020: Market size was $832.8 billion in 2019 - forecast to 2025 - researchandmarkets.com. *Business Wire*. (2021, January 13). Retrieved from <https://www.businesswire.com/news/home/20210113005662/en/Global-Elderly-Care-Products-and-Services-Market-Report-2020-Market-Size-was-832.8-Billion-in-2019---Forecast-to-2025---ResearchAndMarkets.com#:~:text=Elderly%20Care%20Products%20and%20Services%20Market%20top%2010%20companies%20are,Healthcare%20and%20Sunrise%20Senior%20Living>.

HomeSafe standard support. *Lifeline.* (2022, August 22). <https://www.lifeline.com/support/homesafe/>

IOT for seniors: Solutions and use cases. *Cogniteq*. (n.d.). <https://www.cogniteq.com/blog/iot-seniors-solutions-and-use-cases>

JeffB, Mikeljo, Majrooo, ajb2k3, &amp; Maxrom. (2020, January 13). IsCharging() and getbatterylevel(). *M5Stack Community*. <https://community.m5stack.com/topic/1361/ischarging-and-getbatterylevel/4>

M5StickC. *m5Stack*. (n.d.). <http://docs.m5stack.com/en/core/m5stickc>

On the go. *Lifeline.* (2023, March 22). <https://www.lifeline.com/product/on-the-go/>

Schlenker M.S., L., & Hoyt, J. (2023, April 7). Philips Lifeline Medical Alert System Review 2023. *SeniorLiving.org*. <https://www.seniorliving.org/medical-alert-systems/philips-lifeline/reviews/>

Sensor Smoothing. *Sumida Crossing*. (n.d.). <http://www.sumidacrossing.org/LayoutElectricity/microcontrollers/Arduinos/IRSensors/SensorSmoothing/>

# Appendices

## C++ code to Test connection to AWS IoT Service

#include "secrets.h"

#include <WiFiClientSecure.h>

#include <MQTTClient.h>

#include <ArduinoJson.h>

#include "WiFi.h"

#include <M5StickC.h>

#define TFT\_GREY 0x5AEB // New colour

float accX = 0.0F;

float accY = 0.0F;

float accZ = 0.0F;

unsigned long startMillis;

unsigned long currentMillis;

// Publish messages every 10 seconds

 const unsigned long publish\_interval = 10000;

// The MQTT topics that this device should publish/subscribe

#define AWS\_IOT\_PUBLISH\_TOPIC   "esp32/pub"

#define AWS\_IOT\_SUBSCRIBE\_TOPIC "esp32/sub"

WiFiClientSecure net = WiFiClientSecure();

MQTTClient client = MQTTClient(256);

void connectAWS()

{

  WiFi.mode(WIFI\_STA);

  WiFi.begin(WIFI\_SSID, WIFI\_PASSWORD);

  Serial.println("Connecting to Wi-Fi");

  while (WiFi.status() != WL\_CONNECTED){

    delay(500);

    Serial.print(".");

  }

  // Configure WiFiClientSecure to use the AWS IoT device credentials

  net.setCACert(AWS\_CERT\_CA);

  net.setCertificate(AWS\_CERT\_CRT);

  net.setPrivateKey(AWS\_CERT\_PRIVATE);

  // Connect to the MQTT broker on the AWS endpoint we defined earlier

  client.begin(AWS\_IOT\_ENDPOINT, 8883, net);

  // Create a message handler

  client.onMessage(messageHandler);

  Serial.print("Connecting to AWS IOT");

  if (!client.connect("MyNewESP32")) {

    Serial.print(".");

    delay(100);

  }

  if(!client.connected()){

    Serial.println("AWS IoT Timeout!");

    return;

  }

  // Subscribe to a topic

  client.subscribe(AWS\_IOT\_SUBSCRIBE\_TOPIC);

  Serial.println("AWS IoT Connected!");

  M5.Lcd.println("AWS IoT Connected!");

}

void publishMessage()

{

  StaticJsonDocument<200> doc;

  doc["sms"] = "+14804224242";

  doc["message"] = "Help! I've fallen and I can't get up!";

  char jsonBuffer[512];

  serializeJson(doc, jsonBuffer); // print to client

  client.publish(AWS\_IOT\_PUBLISH\_TOPIC, jsonBuffer);

}

void messageHandler(String &topic, String &payload) {

  Serial.println("incoming: " + topic + " - " + payload);

  StaticJsonDocument<200> doc;

  deserializeJson(doc, payload);

  const char\* message = doc["message"];

  M5.Lcd.fillScreen(TFT\_GREY);

  M5.Lcd.setCursor(0, 0, 2);

  M5.Lcd.setTextColor(TFT\_BLUE);

  M5.Lcd.setTextFont(2);

  M5.Lcd.println(message);

}

void setup() {

  //initial start time

  startMillis = millis();

  M5.begin();

  M5.IMU.Init();

  M5.Lcd.setRotation(3);

  M5.Lcd.fillScreen(TFT\_GREY);

  M5.Lcd.setCursor(0, 0, 2);

  M5.Lcd.setTextColor(TFT\_WHITE,TFT\_BLACK);

  M5.Lcd.setTextSize(2);

  Serial.begin(9600);

  connectAWS();

}

void loop() {

  M5.IMU.getAccelData(&accX,&accY,&accZ);

  float at = sqrt(accX\*accX + accY\*accY + accZ\*accZ);

  // Publish a message when a fall is detected

  if(at < 0.10) {

    publishMessage();

    delay(2000);

  }

  client.loop();

}

## C++ Final Project Code

#include <M5StickC.h>

#include <math.h>

#include <vector>

#include "secrets.h"

#include <WiFiClientSecure.h>

#include <MQTTClient.h>

#include <ArduinoJson.h>

#include "WiFi.h"

//Define constants

const double THRESHOLD = 0.3;  //Fall detection threshold in m/s^2

const int WINDOW\_SIZE = 50;    //window size for smoothing data

// The MQTT topics that this device should publish/subscribe

#define AWS\_IOT\_PUBLISH\_TOPIC   "esp32/pub"

#define AWS\_IOT\_SUBSCRIBE\_TOPIC "esp32/sub"

WiFiClientSecure net = WiFiClientSecure();

MQTTClient client = MQTTClient(256);

//Define data structures

struct AccelerometerData {

  float x, y, z;

};

struct GyroscopeData {

  float x, y, z;

};

//Calculate magnitude of acceleration vector

float calculateMagnitude(AccelerometerData data) {

  return sqrt(pow(data.x, 2) + pow(data.y, 2) + pow(data.z, 2));

}

//Smooth Accelerometer data using a sliding window

std::vector<float> smoothData(std::vector<float> data) {

  std::vector<float> smoothedData;

  for (int i = 0; i <= data.size() - WINDOW\_SIZE; i++) {

    float sum = 0;

    for (int j = i; j < i + WINDOW\_SIZE; j++) {

      sum += data[j];

    }

    smoothedData.push\_back(sum / WINDOW\_SIZE);

  }

  return smoothedData;

}

// Function to detect falls based on accelerometer and gyroscope data

bool detectFall(const std::vector<AccelerometerData>& accelerometerData, const std::vector<GyroscopeData>& gyroscopeData) {

  // Calculate standard deviation of the z-axis accelerometer data

  float mean = 0.0;

  for (int i = 0; i < WINDOW\_SIZE; i++) {

    mean += accelerometerData[i].z;

  }

  mean /= WINDOW\_SIZE;

  float stdDev = 0.0;

  for (int i = 0; i < WINDOW\_SIZE; i++) {

    stdDev += pow(accelerometerData[i].z - mean, 2);

  }

  stdDev = sqrt(stdDev / WINDOW\_SIZE);

  // Print standard deviation to serial monitor

  Serial.print("Standard deviation: ");

  Serial.println(stdDev);

  // Detect fall if the standard deviation is above the threshold

  return stdDev > THRESHOLD;

}

void setCursorToLine(int lineNumber) {

  const int X\_LOCATION = 5;

  const int Y\_LOCATION = 0;

  const int LINE\_HEIGHT = 10;

  M5.Lcd.setCursor(X\_LOCATION, Y\_LOCATION + ((lineNumber - 1) \* LINE\_HEIGHT), 2);

}

void printData(const std::vector<AccelerometerData>& accelerometerData, const std::vector<GyroscopeData>& gyroscopeData) {

  Serial.println("Accelerometer data:");

  for (const auto& data : accelerometerData) {

    Serial.printf("X=%.2f, Y=%.2f, Z=%.2f\n", data.x, data.y, data.z);

  }

  Serial.println("Gyroscope data:");

  for (const auto& data : gyroscopeData) {

    Serial.printf("X=%.2f, Y=%.2f, Z=%.2f\n", data.x, data.y, data.z);

  }

}

void connectAWS()

{

  WiFi.mode(WIFI\_STA);

  WiFi.begin(WIFI\_SSID, WIFI\_PASSWORD);

  Serial.println("Connecting to Wi-Fi");

  while (WiFi.status() != WL\_CONNECTED){

    delay(500);

    Serial.print(".");

  }

  // Configure WiFiClientSecure to use the AWS IoT device credentials

  net.setCACert(AWS\_CERT\_CA);

  net.setCertificate(AWS\_CERT\_CRT);

  net.setPrivateKey(AWS\_CERT\_PRIVATE);

  // Connect to the MQTT broker on the AWS endpoint we defined earlier

  client.begin(AWS\_IOT\_ENDPOINT, 8883, net);

  // Create a message handler

  client.onMessage(messageHandler);

  Serial.print("Connecting to AWS IOT");

  while (!client.connect(THINGNAME)) {

    Serial.print(".");

    delay(100);

  }

  if(!client.connected()){

    Serial.println("AWS IoT Timeout!");

    return;

  }

  // Subscribe to a topic

  client.subscribe(AWS\_IOT\_SUBSCRIBE\_TOPIC);

  Serial.println("AWS IoT Connected!");

  M5.Lcd.println("AWS IoT Connected!");

}

void publishMessage()

{

  StaticJsonDocument<200> doc;

  doc["sms"] = "+85265967075";

  doc["message"] = "Help! I've fallen and I can't get up!";

  char jsonBuffer[512];

  serializeJson(doc, jsonBuffer); // print to client

  client.publish(AWS\_IOT\_PUBLISH\_TOPIC, jsonBuffer);

}

void messageHandler(String &topic, String &payload) {

  Serial.println("incoming: " + topic + " - " + payload);

  StaticJsonDocument<200> doc;

  deserializeJson(doc, payload);

  const char\* message = doc["message"];

  M5.Lcd.println(message);

}

void setup() {

  // Initialize the M5StickC object

  M5.begin();

  M5.Lcd.fillScreen(BLACK);

  Serial.begin(115200);

  M5.IMU.Init();

  connectAWS();

}

//Run repeatedly

void loop() {

  // Collect accelerometer and gyroscope data

  std::vector<AccelerometerData> accelerometerData;

  std::vector<GyroscopeData> gyroscopeData;

  for (int i = 0; i < 200; i++) {

    AccelerometerData accelData;

    GyroscopeData gyroData;

    M5.IMU.getAccelData((float\*)&accelData.x, (float\*)&accelData.y, (float\*)&accelData.z);

    M5.IMU.getGyroData((float\*)&gyroData.x, (float\*)&gyroData.y, (float\*)&gyroData.z);

    accelerometerData.push\_back(accelData);

    gyroscopeData.push\_back(gyroData);

    delay(10);

  }

  // Resize the data vectors to the actual number of data points collected

  accelerometerData.resize(200);

  gyroscopeData.resize(200);

  printData(accelerometerData, gyroscopeData);

  //Detect falls

  bool fallDetected = detectFall(accelerometerData, gyroscopeData);

  // Print result

  if (fallDetected) {

    setCursorToLine(3);

    M5.Lcd.printf("Fall detected!");

    Serial.println("Fall detected!");

    publishMessage();

  } else {

    setCursorToLine(3);

    M5.Lcd.printf("No fall detected.");

    Serial.println("No fall detected.");

  }

  // Wait before collecting more data

  delay(1000);

}

# Supervision Log Sheet

|  |  |  |  |
| --- | --- | --- | --- |
| Meeting No. | Date & Time | Duration & Venue | Discussion items / Agreed tasks and action plans for the next meeting |
| 1 | 22/04 | HHB -1706 | Final Report |
| 2 | 23/05 | HHB -1706 | Final Report |
| 3 | 25/04 | HHB -1706 | Final Report |
| 4 | 27/04 | HHB -1706 | Final Report |
| 5 | 28/04 | HHB -1706 | Final Report |