الجمـهوريــة الجزائريــة الديمقراطيــة الشعبيــة
PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA

Ministry of Hgher Education and Scientific Research
Universite Freres Mentouri Constantine 1
Technilogical Sciences Faculty
Electronics Department

وزارة التعليـم العالـي والبحـث العلمـي
جامعــة الإخوة منتوري قسنطينــة 1
كليــة علوم التكنولوجيا
قسم الالكترونيك

Département d'Electronique

MEMOIRE

En vue de l'obtention du diplôme de **LICENCE** (L3-LMD)

Filière : **Automatique**

Thème

Project HALO: Innovating Safety and Efficiency in Construction through Smart Hard Hats.

Encadreur : **M. AMMARI**

Co-encadreur : **L. BENTEROUCHE**

Présenté par :

- DJEBBES Rabah.

Année universitaire : **2023 / 2024**

Work Provided by Student:

- DJEBBES Rabah.

Idea Provided by:

- Prof N. Mansouri.

Acknowledgements:

Allah grant me the serenity to accept the things I cannot change, courage to change the things I can, and the wisdom to know the difference.

Living one day at a time, enjoying one moment at a time, accepting hardships as the pathway to peace.

Taking this world as it is, not as I would have it. Trusting that Allah will make all things right if I Surrender to his will, so that I may be reasonably happy in this life, and supremely happy forever in the next. Amen.

Above all, I would like to say Alhamdulillah "Thank Allah" for giving me the strength and will to complete this modest work. my warm thanks go particularly to Professor N.MANSOURI who proposed this subject and directed my work and her valuable advice and encouragement. In all loyalty, a precious thank you to Professor. M.AMMARI and Professor **L.BENTEROUCHE** for his help and advice. I express my respectful gratitude to all teachers at the University of Constantine 1.

I would also like to thank my classmates. It is with real pleasure that I extend my sincere recognition and deep gratitude to all those who helped me directly or indirectly to carry out this work.

DJEBBES Rabah.

Summary

General Introduction **4**

Chapter (I): **7**

I – 1 - Introduction 8

I – 2 - Project overview 8

I – 3 - Historical contexts 9

I – 4 – Employed methods and techniques 10

I – 5 - Conclusion 10

Chapter (II): **11**

II – 1 - Introduction 12

II – 1 - Components employed 12

II – 1 - Detailed definitions, parameters and examples 12

II – 1 - Conclusion 15

Chapter (III): **16**

III – 1 - Introduction 17

III – 2 - System's functionality 17

a - Key Components and Variables 18

b - Algorithm Steps 19

c - Code Walkthrough 21

d – Summary 23

III – 3 - Obtained Results 23

III – 4 - Conclusion 25

General Conclusion: **26**

Bibliographic References **28**

Complete Code **29**

General Introduction

General Introduction

Construction sites are defined as dynamic environments where productivity and progress go hand in hand with hazards and risks, from intricate structures to towering infrastructures there is no doubt that this industry shapes the environment around us, however safety remains a paramount concern, as construction sites are fraught with dangerous machinery and hazardous tools that can jeopardize the well-being of both workers and bystanders alike.

According to the BLS (U.S Bureau of Labor Statistics), construction has the second most workplace fatalities of all industries, where roughly 1 in every 5 reported deaths is in said industry, as well as reports indicate that fatalities have noticed an 11% increase from 2021 to 2022 where a total of 1,056 workers have died on the job.

Each year, about 1% of construction workers suffer a fatal injury, which is the highest rate in any industry, these injuries are due to what is called the “Fatal Four” leading causes of construction deaths (falls, struck by equipment, caught between objects and electrocutions) which alone account for 65% of all construction-related deaths.

So needless to say, understanding and mitigating these hazards is an essential component for ensuring a safe and secure workplace.

Introducing Project Halo, a smart construction hard hat coupled with an electronic bracelet featured with sensors and instruments to help insure both the well-being of workers as well as a safe and suitable work environment, all of the collected data of the environment measures can be monitored in real time from a central unit, allowing more control and ease of management, while decreasing risks and ensuring fast and efficient emergency dispatch in case of any injuries.

Chapter One 1

Introduction

In modern times data is power, as whoever has access to more valuable information, has access to a treasure trove of opportunities, insights, and advantages.

With current technological advancement and the rise of AI many construction companies have been looking for solutions to track work environment related data, either for analysis purposes or simply to be able to rival competition, some devices were proposed to the market but a whole solution is yet to be found, and the problem remains unsolved.

Project Overview

Project HALO is a groundbreaking initiative aimed to revolutionize workplace efficiency and safety, by integrating advanced sensors and modules, offering real-time monitoring for environmental conditions, ushering in a new era of precision and data driven decision making.

At its core Project HALO harnesses an array of advanced sensors designed to capture a comprehensive spectrum of useful data, from ambient temperature and humidity levels to air quality metrics and GPS coordinates, every aspect of the workplace environment is meticulously monitored, in addition, features such as acceleration and fall detection are integrated to ensure immediate response to critical incidents, further enhancing safety.

The seamless transmission of collected data to a centralized unit via Radio Frequency technology allows for rapid analysis and interpretation, by leveraging the provided information, monitors can identify potential risks in real-time enabling risk mitigation and proactive intervention strategies. Furthermore, the complete elimination of human error, guesswork and the reliance on outdated practices all of which to enhance operational efficiency and to minimize errors.

In essence, this prototype represents a paradigm shift in workplace safety monitoring, empowering companies to keep their workers safe while optimizing productivity, paving the way for safer, smarter and more sustainable practices.

Historical Context

The evolution of hard hats can be dated back to the early 20th century when a surge was induced in industrialization and construction activities, requiring improved safety measures for workers. The first modern hard hat, known as the "Hard Boiled Hat," was introduced by Bullard in 1919. It was made of steamed canvas, glue, and black paint, providing rudimentary protection against falling debris and impacts.

Over the decades, advancements in manufacturing processes and materials science led to the development of a hard hats made from more durable and lightweight materials such as fiberglass, thermoplastics, and high-density polyethylene (HDPE). These improvements significantly enhanced the protective capabilities of the hard hats while also increasing comfort.

In recent years, with the advent of the Internet of Things (IoT) and wearable technology, traditional hard hats have undergone a transformative shift towards becoming more "smart" personal protective equipment (PPE). Smart hard hats integrate various sensors, communication modules, and computing capabilities to offer features beyond basic shock and impact protection.

These modern innovations enable smart hard hats to provide real-time monitoring of environmental conditions such as temperature, humidity, and air quality, thereby alerting workers to potential hazards. They can also incorporate augmented reality (AR) displays, enabling workers to access vital information hands-free, such as blueprints, safety protocols, and real-time instructions.

Furthermore, smart hard hats can enhance communication and coordination among workers through built-in radios, Bluetooth connectivity, and even integration with existing project management systems. This fosters greater collaboration, efficiency, and safety on construction sites and in industrial settings.

Overall, the evolution of hard hats into smart wearable devices represents a significant advancement in workplace safety and productivity, aligning with the ongoing pursuit of technological innovation to mitigate risks and improve the well-being of workers in hazardous environments.

Employed methods and techniques

In the creation of the smart hard hats, a combination of advanced methods and techniques is employed to ensure optimal functionality and safety. Leveraging principles from materials science, engineers utilize cutting-edge materials such as impact-resistant thermoplastics and lightweight composites to craft hard hat shells that provide robust protection without compromising wearer comfort. Integration of sensor technologies, including accelerometers, gyroscopes, and environmental sensors, enables real-time monitoring of vital parameters such as temperature, humidity, and air quality, empowering workers to stay informed of potential hazards.

Through the strategic application of these methods and techniques, smart hard hats stand at the forefront of innovation, safeguarding workers while fostering efficiency in diverse industrial and construction environments.

Conclusion

In conclusion, the evolution of smart hard hats represents a remarkable fusion of cutting-edge engineering and innovative technology, aimed at revolutionizing workplace safety and productivity. Through the utilization of advanced materials, sensor technologies, these intelligent PPE solutions offer comprehensive protection while empowering workers with real-time information and enhanced communication capabilities. As a result, smart hard hats not only mitigate risks in hazardous environments but also streamline workflows and foster collaboration among team members. With a commitment to ongoing research and development, the future holds even greater potential for smart hard hats to continue driving advancements in safety, efficiency, and well-being across various industries.

Chapter Two 2

Introduction

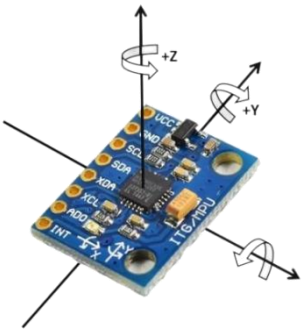
In an era marked by rapid technological advancement, the evolution of personal protective equipment (PPE) has witnessed a transformative shift towards smart solutions. Among these innovations, smart hard hats stand out as a beacon of progress, embodying the fusion of traditional safety gear with cutting-edge technology. This chapter endeavors to delve into the intricate world of smart hard hats, unraveling their multifaceted functionalities, and examining their profound implications for workplace safety and efficiency.

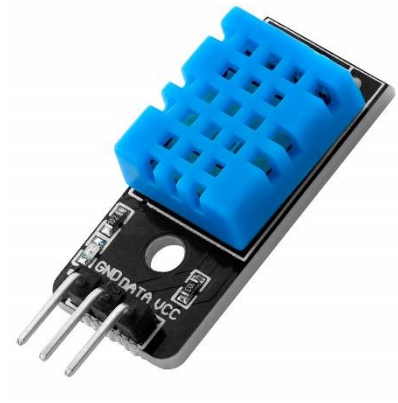
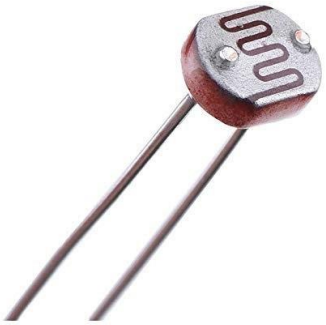
Components Employed

At the core of every smart hard hat lies a sophisticated array of components meticulously engineered to safeguard workers while enhancing their operational capabilities. These components encompass a diverse range of sensors, communication modules, computing units, and display interfaces. The hard hat shell itself is often crafted from advanced materials such as impact-resistant thermoplastics or lightweight composites, striking an optimal balance between durability and wearer comfort. Embedded within this shell are an array of sensors, including accelerometers, gyroscopes, temperature sensors, and environmental monitors, which continuously monitor the wearer's surroundings. Additionally, smart hard hats may feature integrated communication modules, such as WIFI and BLE, enabling seamless coordination among team members.

Modules Used

To fully grasp the intricacies of smart hard hats, it is essential to delve into detailed definitions, parameters, and real-world examples showcasing their capabilities. One fundamental aspect of smart hard hats is their ability to provide real-time monitoring of environmental conditions. For instance, temperature sensors embedded within the hard hat can alert workers to extreme heat conditions, prompting them to take necessary precautions. Similarly, humidity sensors can warn of elevated moisture levels, which may indicate the presence of mold or other hazards.

 **II – 1 - MPU 6050**	MPU6050 sensor module is complete 6-axis Motion Tracking Device. It combines 3-axis Gyroscope, 3-axis Accelerometer and Digital Motion Processor all in small package. Also, it has additional feature of on-chip Temperature sensor. It has I2C bus interface to communicate with the microcontrollers. It has Auxiliary I2C bus to communicate with other sensor devices like 3-axis Magnetometer, Pressure sensor etc. If 3-axis Magnetometer is connected to auxiliary I2C bus, then MPU6050 can provide complete 9-axis Motion Fusion output.
 **II – 2 - NEO 6m**	The NEO-6M is a compact and versatile Global Navigation Satellite System (GNSS) module manufactured by u-blox. It is designed to provide accurate positioning data by receiving signals from multiple satellite constellations, including GPS (Global Positioning System), GLONASS (Global Navigation Satellite System), Galileo, and BeiDou. The NEO-6M module features a built-in antenna and receiver, making it easy to integrate into various applications requiring precise location information, such as navigation systems, vehicle tracking devices, drones, and asset management solutions. With its small form factor, low power consumption, and high sensitivity, the NEO-6M is widely used across industries where reliable positioning is essential.

 **II – 3 - DHT 11**	DHT11 Temperature & Humidity Sensor features a temperature & humidity sensor complex with a calibrated digital signal output. By using the exclusive digital-signal-acquisition technique and temperature & humidity sensing technology, it ensures high reliability and excellent long-term stability. This sensor includes a resistive-type humidity measurement component and an NTC temperature measurement component, and connects to a high-performance 8-bit microcontroller, offering excellent quality, fast response, anti-interference ability and cost-effectiveness.
 **II – 4 - LDR**	LDR (Light Dependent Resistor) as the name states is a special type of resistor that works on the photoconductivity principle means that resistance changes according to the intensity of light. Its resistance decreases with an increase in the intensity of light. It is often used as a light sensor, light meter, Automatic Street light, and in areas where we need to have light sensitivity. LDR is also known as a Light Sensor.
 **II – 5 - MQ135**	An MQ135 air quality sensor is one type of MQ gas sensor used to detect, measure, and monitor a wide range of gases present in air like ammonia, alcohol, benzene, smoke, carbon dioxide, etc. It is a semiconductor air quality check sensor suitable for monitoring applications of air quality. It is highly sensitive to NH3, NOx, CO2, benzene, smoke, and other dangerous gases in the atmosphere. It is available at a low cost for harmful gas detection and monitoring applications.

Conclusion

In conclusion, it's crucial to recognize that the bulky modules described in this exploration were primarily utilized for prototyping purposes, serving as proof of concept for the potential of smart hard hats. However, in the context of commercialization, a significant emphasis would be placed on optimizing hardware to achieve a delicate balance between cost-effectiveness, wearability, and comfort. This optimization process would likely involve the integration of more streamlined and compact components, aimed at reducing production costs while enhancing the overall user experience. By prioritizing these factors, the transition from prototype to commercial product would not only make smart hard hats more accessible to a wider range of industries but also ensure that they seamlessly integrate into existing work environments without sacrificing safety or convenience.

Chapter Three 3

Introduction

As we delve deeper into the realm of smart hard hats, this chapter aims to provide a comprehensive overview of the implementation process, detailing the functionality, components, simulations, trials, and results obtained during the development and testing phases.

System's Functionality

The primary functionality of the smart hard hat, Project HALO, revolves around its ability to continuously monitor the environmental conditions. The system is designed to detect critical parameters such as temperature, humidity, air quality, and worker movements. Here are the key functionalities:

1 - Environmental Monitoring: Sensors embedded in the hard hat measure temperature, humidity, and air quality, alerting workers to potential hazards such as extreme heat or the presence of harmful gases.

2 - Fall Detection: Accelerometers and gyroscopes integrated into the hard hat detect sudden movements indicative of falls. In the event of a fall, an immediate alert is sent to the central monitoring unit.

3 - Real-Time Data Transmission: All collected data is transmitted in real-time to a central monitoring unit using WIFI technology. This allows for continuous oversight and rapid response to any detected issues as well as provides an unlimited range.

Fall Detection Algorithm Explanation

The fall detection algorithm implemented in the provided code uses the MPU-6050 sensor to monitor the acceleration and gyroscope data to detect potential falls. The process involves several steps and multiple triggers to determine if a fall has occurred. Below is a detailed explanation of the fall detection logic.

Key Components and Variables:

1 - MPU-6050 Sensor:

A sensor that provides accelerometer and gyroscope data.

2 - Accelerometer Data:

Measures the acceleration forces along the X, Y, and Z axes.

3 - Gyroscope Data:

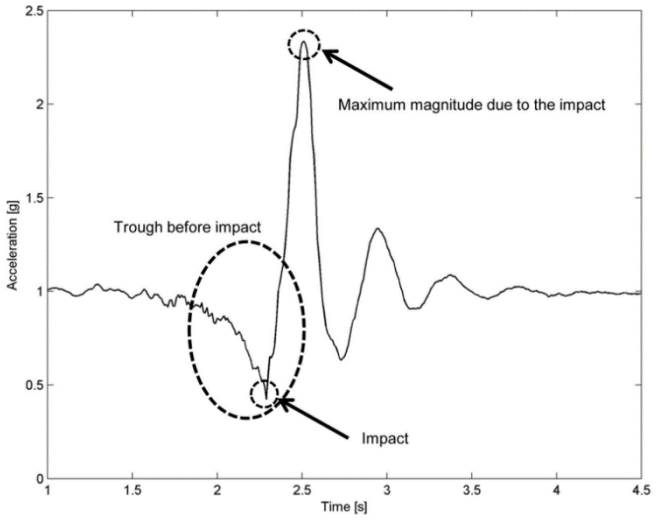
Measures the rotational velocity along the X, Y, and Z axes.

4 - Thresholds and Triggers:

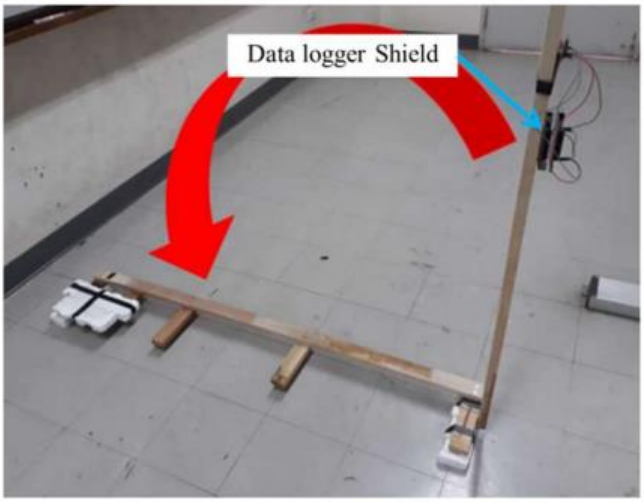
- trigger1: Activated when the acceleration amplitude falls below a certain lower threshold.
- trigger2: Activated when the acceleration amplitude exceeds an upper threshold after trigger1 is activated.
- trigger3: Activated when a significant change in orientation is detected after trigger2 is activated.

5 - Counters:

trigger1count, trigger2count, trigger3count: Track the number of cycles since each respective trigger was activated.



III – 1 – 1



III – 1 - 2

Algorithm Steps:

1 Reading Sensor Data:

a. The "mpu_read()" function reads the accelerometer and gyroscope data from the MPU-6050 sensor.

2 Calculating Amplitude Vector:

a. The raw acceleration data is processed to calculate the amplitude vector (Raw_Amp), which is a measure of the overall acceleration magnitude. This is done using the formula:

$$Raw\_Amp = \sqrt{ax^2 + ay^2 + az^2}$$

The amplitude is then scaled and stored in "Amp".

3 Trigger 1 : Lower Threshold Check :

a. If the amplitude (Amp) is less than or equal to 2 and trigger2 is not active, trigger1 is set to true.

b. This indicates a significant decrease in acceleration, potentially the start of a fall.

4 Trigger 2 : Upper Threshold Check :

a. If trigger1 is active, the "trigger1count" is incremented.

b. If the amplitude exceeds 12, "trigger2" is activated, "trigger1" is reset, and "trigger1count" is reset.

c. This step checks for a sudden increase in acceleration following the initial drop, characteristic of the impact during a fall.

5 Trigger 3 : Orientation Change Check :

a. If “trigger2“ is active, the “trigger2count” is incremented.

b. The algorithm calculates the change in orientation using the gyroscope data:

$$angleChange = \sqrt{gx^2 + gy^2 + gz^2}$$

c. If the “angleChange” falls within a certain range (30 to 400), “trigger3” is activated, “trigger2” is reset, and “trigger2count” is reset.

d. This step checks for a significant change in orientation, indicating that the person has fallen and is in an unusual position.

6 Fall Confirmation :

a. If “trigger3” is active, the “trigger3count” is incremented.

b. After 10 cycles of “trigger3” being active, the “angleChange” is recalculated.

c. If the “angleChange” is within 0 to 10, indicating no significant movement, it confirms the fall, and the fall flag is set to true.

d. If the orientation returns to normal, “trigger3” is reset.

7 Fall Detected :

a. If a fall is confirmed (fall is true), an SOS function “SOS()” is called, which triggers an alert.

b. The “fall” flag is then reset.

8 Resetting Triggers :

a. If “trigger2” is active for more than 6 cycles without progressing to “trigger3”, it is reset.

b. Similarly, if “trigger1” is active for more than 6 cycles without progressing to “trigger2”, it is reset.

Code Walkthrough:

Here's the relevant code for the fall detection algorithm:

```
void mpu_check() {
  mpu_read(); // Read MPU data
  ax = (AcX - 2050) / 16384.00;
  ay = (AcY - 77) / 16384.00;
  az = (AcZ - 1947) / 16384.00;
  gx = (GyX + 270) / 131.07;
  gy = (GyY - 351) / 131.07;
  gz = (GyZ + 136) / 131.07;

  // Calculate amplitude vector for 3 axes
  float Raw_Amp = pow(pow(ax, 2) + pow(ay, 2) + pow(az, 2), 0.5);
  int Amp = Raw_Amp * 10;  // Scaled amplitude
  Blynk.virtualWrite(V4, Raw_Amp);
  Serial.println(Amp);

  // Trigger 1: Lower threshold
  if (Amp <= 2 && !trigger2) {
    trigger1 = true;
    Serial.println("TRIGGER 1 ACTIVATED");
  }

  // Trigger 2: Upper threshold
  if (trigger1) {
    trigger1count++;
    if (Amp >= 12) {
      trigger2 = true;
      Serial.println("TRIGGER 2 ACTIVATED");
      trigger1 = false;
      trigger1count = 0;
    }
  }

  // Trigger 3: Orientation change
  if (trigger2) {
    trigger2count++;
    angleChange = pow(pow(gx, 2) + pow(gy, 2) + pow(gz, 2), 0.5);
    Serial.println(angleChange);
    if (angleChange >= 30 && angleChange <= 400) {
      trigger3 = true;
      trigger2 = false;
      trigger2count = 0;
      Serial.println("TRIGGER 3 ACTIVATED");
    }
```

```
  }

  // Fall confirmation
  if (trigger3) {
    trigger3count++;
    if (trigger3count >= 10) {



      angleChange = pow(pow(gx, 2) + pow(gy, 2) + pow(gz, 2), 0.5);
      Serial.println(angleChange);
      if (angleChange >= 0 && angleChange <= 10) {
        fall = true;
        trigger3 = false;
        trigger3count = 0;
        Serial.println("FALL DETECTED");
      } else {
        trigger3 = false;
        trigger3count = 0;
        Serial.println("TRIGGER 3 DEACTIVATED");
      }
    }
  }

  // Fall detected action
  if (fall) {
    Serial.println("FALL DETECTED");
    SOS();
    fall = false;
  }

  // Reset triggers if counts exceed limits
  if (trigger2count >= 6) {
    trigger2 = false;
    trigger2count = 0;
    Serial.println("TRIGGER 2 DEACTIVATED");
  }
  if (trigger1count >= 6) {
    trigger1 = false;
    trigger1count = 0;
    Serial.println("TRIGGER 1 DEACTIVATED");
  }
  delay(100); // Delay for sampling rate
}
```

Summary:

The fall detection algorithm relies on a sequence of triggers to confirm a fall event. It monitors the amplitude of acceleration and changes in orientation to detect sudden impacts and unusual postures indicative of a fall. If these conditions are met, the algorithm activates an SOS alert, ensuring rapid response to potential accidents. The careful sequence of checks helps reduce false positives, ensuring that only genuine falls trigger alerts.

Obtained Results

To evaluate the performance of the proposed system, six types of activity experiments were conducted, including one fall and five activities representative of Active Daily Lives (ADLs), namely walking, standing-sitting down in a chair, standing-sitting down on the floor, sitting-lying on the floor, and walking up-down the stairs. Each action was performed 100 times. And thus, 600 test samples were collected. The device was set up on the tester's head.

Performance of the falling detection system was analyzed by accuracy, sensitivity and specificity, which can be calculated from four possible outcomes, which include true positive (TP), true negative (TN), false positive (FP), and false negative (FN), as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Sensitivity = \frac{TP}{TP + FN}$$

$$Specificity = \frac{TN}{TN + FP}$$

In these equations, true positive (TP) is defined as an event in which the device produced an alert when a fall was detected.

True negative (TN) is defined as an event in which the device did not produce an alert when a fall was not detected.

False positive (FP) is defined as an event in which the device produced an alert, but a fall

was not detected.

False negative (FN) is defined as an event in which the device did not produce an alert, but a fall was detected.

Table 1. The Sensitivity, specificity, and accuracy of the proposed system.

	Activity experiments	Number of experiments	True positive (TP)	False negative (FN)	True negative (TN)	False positive (FP)	Performance
Sensitivity	Falling	100	88	12			88%
Specificity	Walking	100			96	4	98.4%
	Standing-sitting down in a chair	100			99	1	
	Standing-sitting down on the floor	100			99	1	
	Sitting-lying on the floor	100			100	0	
	Walking up-down the stairs	100			98	2	
Accuracy		600	88	12	492	8	96.7%

The results of the experimental data are shown in table 1. A total of 88 out of 100 fall actions triggered the alarm, therefore the sensitivity was 88%. Eight out of 492 ADLs triggered the alarm; thus, the specificity was 98.4%. The accuracy of the proposed system is 96.7%, which is very high. A total of 12 out of 100 non-triggering falls occurred because the falling acceleration was not below the threshold. The eight out of 500 triggering falls occurred because the tester moved too fast during the activity experiments, and the system mistook this for a falling action.

Conclusion:

In this work, we presented the design and development of a fall detection system using the ESP32 microcontroller. This system can send alerts to a dashboard via a WIFI network, chosen for its speed, reliability, and compatibility with Industry 4.0 and the growing wave of IoT devices. The system's accuracy is notably high at 96.7%. This prototype is both highly effective and cost-efficient.

The impetus for this project stems from the increasing need for advanced monitoring solutions in dynamic and hazardous environments, such as construction sites. By leveraging WIFI technology, the system ensures fast and reliable data transmission, which is crucial for real-time monitoring and rapid response in case of falls.

The core functionality of the fall detection system is enabled by the MPU-6050 sensor, which monitors acceleration and gyroscope data to detect falls. The fall detection algorithm accurately differentiates between falls and other activities of daily living (ADLs) through a series of threshold-based triggers, minimizing false positives and ensuring genuine falls are detected.

In extensive testing, the system demonstrated high sensitivity and specificity, further validating its effectiveness. This high accuracy, combined with its low cost, makes the prototype a promising solution for enhancing workplace safety.

General Conclusion

Conclusion

In the dynamic landscape of modern workplaces, ensuring the safety and well-being of workers is paramount. Amidst this imperative, emerging technologies have catalyzed a paradigm shift in occupational safety, giving rise to innovative solutions aimed at mitigating risks and enhancing efficiency.

Project HALO stands as a testament to this ethos, representing a groundbreaking initiative poised to revolutionize workplace safety through the convergence of advanced sensors, real-time monitoring capabilities, and proactive intervention strategies. By harnessing the power of smart technology, Project HALO transcends traditional safety paradigms, ushering in a new era of precision, data-driven decision-making, and unparalleled protection for workers in diverse industrial settings. This chapter embarks on a comprehensive journey through the development, implementation, and impact of Project HALO, delving into its multifaceted functionalities, robust algorithms, and tangible outcomes that underscore its pivotal role in fostering safer, smarter, and more sustainable work environments.

Overall, this project represents a significant step forward in integrating IoT technology with personal protective equipment, aligning with the ongoing evolution of smart devices aimed at improving safety and efficiency in various industries. Future research and development will focus on refining the system and exploring additional applications within the realm of smart wearable technology.

Bibliographic References

[1] Census of Fatal Occupational Injuries News Release "USDL-23-2615" released 10:00 a.m. (ET) Tuesday, December 19, 2023 ' https://www.bls.gov/news.release/cfoi.htm '.

[2] William Harris, MS, Thomas Yohannes, MPH, Amber Brooke Trueblood, DrPH1 "Fatal and Nonfatal Focus Four Injuries in Construction" released march 2023 ' https://www.cpwr.com/wp-content/uploads/DataBulletin-March2023.pdf '.

[3] The History of the Hard Hat by "David Kindy" released on February 21, 2020 ' https://www.smithsonianmag.com/innovation/history-hard-hat-180974238/ '

[4] Nguyen Gia T et al 2018 Microprocess. Microsyst. 56 34 ' https://publik.tuwien.ac.at/files/publik_266882.pdf '

[5] Wu Y, Su Y, Feng R, Yu N and Zang X 2019 Measurement 140 283 ' https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9824604/ '

[6] Mastorakis G, Ellis T and Makris D 2018 Expert Syst. Appl. 112 125 ' https://www.sciencedirect.com/science/article/abs/pii/S0957417418303658 '

[7] Bosch-Jorge M, Sánchez-Salmerón A-J, Valera Á and Ricolfe-Viala C 2014 Expert Syst. Appl. 41 7980 ' https://www.sciencedirect.com/science/article/abs/pii/S0957417414004011 '

[8] Tran T-H, Le T-L, Hoang V-N and Vu H 2017 Comput. Methods Programs Biomed. 146 151 ' https://www.mica.edu.vn/perso/Tran-Thi-Thanh-Hai/Pdf/2017_CMPB.pdf '

[9] Peng Y, Peng J, Li J, Yan P and Hu B 2019 Procedia Comput. Sci. 147 27 ' https://ev.fe.uni-lj.si/3-2020/Peng.pdf '

[10] Last Minute Engineers ' https://lastminuteengineers.com/ '

Complete Code

```
#define BLYNK_TEMPLATE_ID "TMPL2wHLuufE4"
#define BLYNK_TEMPLATE_NAME "Project HALO"
#define BLYNK_AUTH_TOKEN "YRRmrSPuBGWS2aoArmCyvwFS8JPrapC1"

#include <BlynkSimpleEsp32.h>
#include <TinyGPSPlus.h>
#include <Wire.h>
#include <SoftwareSerial.h>
#include <DHT.h>

#define DHTPIN 32
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);

TinyGPSPlus gps;

int light_threshhold = 50;
int forced_light = 0;
int particles_threshhold = 70;
const int baud_rate = 115200;

const int smokeSensor = 34;
const int light = 25;
const int light_sensor = 35;
const int SOS_pin = 23;
const int alarm_pin = 13;
const int warn_light = 19;
const int danger_light = 18;

float humidity, temperature;
int light_level, particles;
bool iot = 0;

const int MPU_addr = 0x68; // Define the I2C address of the MPU-6050
int16_t AcX, AcY, AcZ, Tmp, GyX, GyY, GyZ; // Variables to store sensor
data
float ax = 0, ay = 0, az = 0, gx = 0, gy = 0, gz = 0; // Variables to
store processed sensor data
boolean fall = false; // Variable to store if a fall has occurred
boolean trigger1 = false; // Variable to store if first trigger (lower
threshold) has occurred
```

```
boolean trigger2 = false; // Variable to store if second trigger (upper threshold) has occurred
boolean trigger3 = false; // Variable to store if third trigger (orientation change) has occurred
byte trigger1count = 0; // Counter for counts past since trigger 1 was set true
byte trigger2count = 0; // Counter for counts past since trigger 2 was set true
byte trigger3count = 0; // Counter for counts past since trigger 3 was set true
int angleChange = 0; // Variable to store angle change

struct coordinate {
    double lat=0;
    double lng=0;
};

void setup() {
  Serial.begin(baud_rate);

  Serial2.begin(9600); // Initialize serial communication for NEO 6m

  Wire.begin(); // Initialize I2C communication
  Wire.beginTransmission(MPU_addr); // Begin transmission to MPU-6050
  Wire.write(0x6B); // Write to the PWR_MGMT_1 register
  Wire.write(0); // Set register value to zero to wake up the MPU-6050
  Wire.endTransmission(true); // End transmission
  Serial.println("Wrote to IMU"); // Print message to serial monitor

  dht.begin(); // Initialize serial communication with DHT11

  // Pins Setup :
  pinMode(light, OUTPUT);
  pinMode(warn_light, OUTPUT);
  pinMode(danger_light, OUTPUT);
  pinMode(alarm_pin, OUTPUT);
  pinMode(smokeSensor , INPUT);
  pinMode(light_sensor, INPUT);
  pinMode(SOS_pin, INPUT);


  Wake_up(); // Wake up sound
}
```

```
void Wake_up(){
  Blynk.virtualWrite(V11, 0);

  digitalWrite(alarm_pin, HIGH);
  delay(100);
  digitalWrite(alarm_pin, LOW);
  delay(25);
  for (int i=0; i<=3; i++){
    digitalWrite(alarm_pin, HIGH);
    delay(50);
    digitalWrite(alarm_pin, LOW);
    delay(12);
  }
}


void setAlarm(String type="w"){
  if (type=="w"){
    digitalWrite(warn_light, HIGH);
    if(iot){Blynk.virtualWrite(V12, 1);}
    for (int i=0; i<10; i++){
      digitalWrite(alarm_pin, HIGH);
      delay(45);
      digitalWrite(alarm_pin, LOW);
      delay(5);
    }
  } else if (type=="d"){
    digitalWrite(danger_light, HIGH);
    if(iot){Blynk.virtualWrite(V13, 1);}
    for (int i=0; i<10; i++){
      digitalWrite(alarm_pin, HIGH);
      delay(200);
      digitalWrite(alarm_pin, LOW);
      delay(50);
    }
  }
  digitalWrite(warn_light, LOW);
  digitalWrite(danger_light, LOW);
  if(iot){Blynk.virtualWrite(V12, 0);}
  if(iot){Blynk.virtualWrite(V13, 0);}
}
```

```
void SOS() {
  if (iot){Blynk.virtualWrite(V11, 1);}
  while(1){
    setAlarm("d");
    int status = digitalRead(SOS_pin);
    if (status){
      delay(3000);
      status = digitalRead(SOS_pin);
      if (status){
        if (iot){Blynk.virtualWrite(V11, 0);}
        setAlarm("w");
        break;
      }
    }
  }

}


void SOS_check(){
  int status = digitalRead(SOS_pin);
  if (status){
    delay(3000);
    status = digitalRead(SOS_pin);
    if (status){
      SOS();
    }else {
      iot = !iot;
      Serial.print("IOT mode set to : ");
      Serial.println(iot);
    }
  }
}


void check_light(bool debug=0){
  light_level = map(analogRead(light_sensor), 2500, 4096, 0, 100);
  if (iot){Blynk.virtualWrite(V3, light_level);}
  if (debug){
    Serial.print("# Light level >> ");
    Serial.println(light_level);
    Serial.print(" %");
  }
  if(!forced_light){
```

```
    if (light_level <= light_threshhold) {
      digitalWrite(light, HIGH);
      dacWrite(light, 255);
      if (iot){Blynk.virtualWrite(V10, 1);}
    } else {
      digitalWrite(light, LOW);
      dacWrite(light, 0);
      if (iot){Blynk.virtualWrite(V10, 0);}
    }
  }
}


void dht_get_data(bool debug=0){
  humidity = dht.readHumidity();
  temperature = dht.readTemperature();
  if (isnan(humidity) || isnan(temperature)) {
    if(iot){
      Blynk.virtualWrite(V0, 0);
      Blynk.virtualWrite(V1, 0);
    }
  } else {
    if(iot){
      Blynk.virtualWrite(V0, temperature);
      Blynk.virtualWrite(V1, humidity);
    }
  }
  if (debug){
    Serial.print("# Humidity >> ");
    Serial.print(humidity);
    Serial.println("%");
    Serial.print("# Temperature >> ");
    Serial.print(temperature);
    Serial.println("°C ");
  }


}


void check_air(bool debug=0){
  particles = map(analogRead(smokeSensor), 1, 4096, 100, 0);
  if (iot){Blynk.virtualWrite(V2, particles);}
```

```
  if (particles < particles_threshhold){setAlarm("w");}
  if (debug){
    Serial.print("# Air particles >> ");
    Serial.println(particles);
    Serial.print(" %");
  }
}


coordinate get_gps_coordinates(bool debug=0, int retries=1){
  coordinate x;
  int tries = 0;
  while(!x.lat || !x.lng){
    if (Serial2.available() > 0) {
      if (gps.encode(Serial2.read())) {
        if (gps.location.isValid()) {
          x.lat = gps.location.lat();
          Blynk.virtualWrite(V5, x.lat);
          x.lng = gps.location.lng();
          Blynk.virtualWrite(V6, x.lng);
          if (debug){
            Serial.println("---------- gps -----------");
            Serial.print(F("# latitude >> "));
            Serial.println(gps.location.lat());

            Serial.print(F("# longitude >> "));
            Serial.println(gps.location.lng());
            Serial.println("--------------------------");
          }
          break;
        } else if(tries > retries) {
          break;
        } else {
          tries += 1;
          Serial.println(F("# INVALID location !"));
          Blynk.virtualWrite(V5, 0.000);
          Blynk.virtualWrite(V6, 0.000);
          delay(300);
        }
      }
    }
  }
  return x;
}
```

```
void mpu_check(){
  mpu_read();
  ax = (AcX - 2050) / 16384.00;
  ay = (AcY - 77) / 16384.00;
  az = (AcZ - 1947) / 16384.00;
  gx = (GyX + 270) / 131.07;
  gy = (GyY - 351) / 131.07;
  gz = (GyZ + 136) / 131.07;
  // calculating Amplitute vactor for 3 axis
  float Raw_Amp = pow(pow(ax, 2) + pow(ay, 2) + pow(az, 2), 0.5);
  int Amp = Raw_Amp * 10;  // Mulitiplied by 10 bcz values are between
0 to 1
  if (iot){Blynk.virtualWrite(V4, Raw_Amp);}
  //Serial.println(Amp);

  // Check for trigger conditions
  if (Amp <= 2 && trigger2 == false) { // If amplitude breaks lower
threshold
    trigger1 = true; // Set first trigger to true
    Serial.println("TRIGGER 1 ACTIVATED"); // Print trigger message
  }
  if (trigger1 == true) { // If first trigger is true
    trigger1count++; // Increment trigger 1 count
    if (Amp >= 12) { // If amplitude breaks upper threshold
      trigger2 = true; // Set second trigger to true
      Serial.println("TRIGGER 2 ACTIVATED"); // Print trigger message
      trigger1 = false; // Reset first trigger
      trigger1count = 0; // Reset trigger 1 count
    }
  }
  if (trigger2 == true) { // If second trigger is true
    trigger2count++; // Increment trigger 2 count
    angleChange = pow(pow(gx, 2) + pow(gy, 2) + pow(gz, 2), 0.5); //
Calculate angle change
    Serial.println(angleChange); // Print angle change to serial
monitor
    if (angleChange >= 30 && angleChange <= 400) { // If orientation
changes by 80-100 degrees
      trigger3 = true; // Set third trigger to true
      trigger2 = false; // Reset second trigger
      trigger2count = 0; // Reset trigger 2 count
      Serial.println("TRIGGER 3 ACTIVATED"); // Print trigger message
```

```
    }
  }
  if (trigger3 == true) { // If third trigger is true
    trigger3count++; // Increment trigger 3 count
    if (trigger3count >= 10) { // After 10 counts
      angleChange = pow(pow(gx, 2) + pow(gy, 2) + pow(gz, 2), 0.5); //
Recalculate angle change
      Serial.println(angleChange); // Print angle change to serial
monitor
      if ((angleChange >= 0) && (angleChange <= 10)) { // If
orientation remains between 0-10 degrees
        fall = true; // Set fall flag to true
        trigger3 = false; // Reset third trigger
        trigger3count = 0; // Reset trigger 3 count
        Serial.println("FALL DETECTED"); // Print fall detection
message
      } else { // If user regained normal orientation
        trigger3 = false; // Reset third trigger
        trigger3count = 0; // Reset trigger 3 count
        Serial.println("TRIGGER 3 DEACTIVATED"); // Print trigger
message
      }
    }
  }
  if (fall == true) { // If fall is detected
    Serial.println("FALL DETECTED"); // Print fall detection message
    SOS();
    fall = false; // Reset fall flag
  }
  if (trigger2count >= 6) { // If more than 6 counts have passed since
trigger 2
    trigger2 = false; // Reset second trigger
    trigger2count = 0; // Reset trigger 2 count
    Serial.println("TRIGGER 2 DEACTIVATED"); // Print trigger message
  }
  if (trigger1count >= 6) { // If more than 6 counts have passed since
trigger 1
    trigger1 = false; // Reset first trigger
    trigger1count = 0; // Reset trigger 1 count
    Serial.println("TRIGGER 1 DEACTIVATED"); // Print trigger message
  }
  delay(100); // Delay for sampling rate
}
```

```
void mpu_read() {
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x3B);  // starting with register 0x3B (ACCEL_XOUT_H)
  Wire.endTransmission(false);
  Wire.requestFrom(MPU_addr, 14, true); // request a total of 14
registers
  AcX = Wire.read() << 8 | Wire.read(); // 0x3B (ACCEL_XOUT_H) & 0x3C
(ACCEL_XOUT_L)
  AcY = Wire.read() << 8 | Wire.read(); // 0x3D (ACCEL_YOUT_H) & 0x3E
(ACCEL_YOUT_L)
  AcZ = Wire.read() << 8 | Wire.read(); // 0x3F (ACCEL_ZOUT_H) & 0x40
(ACCEL_ZOUT_L)
  Tmp = Wire.read() << 8 | Wire.read(); // 0x41 (TEMP_OUT_H) & 0x42
(TEMP_OUT_L)
  GyX = Wire.read() << 8 | Wire.read(); // 0x43 (GYRO_XOUT_H) & 0x44
(GYRO_XOUT_L)
  GyY = Wire.read() << 8 | Wire.read(); // 0x45 (GYRO_YOUT_H) & 0x46
(GYRO_YOUT_L)
  GyZ = Wire.read() << 8 | Wire.read(); // 0x47 (GYRO_ZOUT_H) & 0x48
(GYRO_ZOUT_L)
}


void loop() {
  //if (!Blynk.connected()){Blynk.begin(BLYNK_AUTH_TOKEN, "Project
Halo", "02022002", );} //"Ooredoo 4G_Work.", "wtrn@9271"
  if (iot) {
    if (!WiFi.isConnected()) {
        Serial.println("Wi-Fi not connected. Attempting to
connect...");
        Blynk.connectWiFi("Project Halo", "02022002");
        Serial.println("Wi-Fi connection initiated.");
    } else {
        Serial.println("Wi-Fi already connected.");
    }

    // Check Blynk connection status
    if (!Blynk.connected()) {
        Serial.println("Blynk not connected. Attempting to
reconnect...");
        Blynk.config(BLYNK_AUTH_TOKEN);
        if (Blynk.connect()) {
```

```
            Serial.println("Blynk reconnected.");
        } else {
            Serial.println("Failed to reconnect to Blynk server.");
            // Handle Blynk reconnection failure
        }
    } else {
        Serial.println("Blynk already connected.");
    }

    Serial.print("IOT mode set to : ");
    Serial.println(iot);
  } else if (Blynk.connected()) {
      Serial.println("Disconnecting from Blynk and turning off Wi-
Fi...");
      ESP.restart();
  }


  get_gps_coordinates();
  dht_get_data();
  check_light();
  check_air();
  mpu_check();
  SOS_check();
  delay(250);
}


BLYNK_WRITE(V9) // Executes when the value of virtual pin 5 changes
  {
    light_threshhold = param.asInt();
    forced_light = 0;
    Serial.println("# Light Threshold Set to : ");
    Serial.println(light_threshhold);
}


BLYNK_WRITE(V7) // Executes when the value of virtual pin 5 changes
  {
    particles_threshhold = param.asInt();
    Serial.println("# Air Quality Threshold Set to : ");
    Serial.println(particles_threshhold);
}
```

```
BLYNK_WRITE(V10) // Executes when the value of virtual pin 5 changes
  {
    int light_status = param.asInt();
    forced_light = 1;
    if (light_status == 1) {
      digitalWrite(light, HIGH);
      dacWrite(light, 255);
      Serial.println("# Light is ON");
      delay(100);
      Blynk.virtualWrite(V10, 1);
    } else {
      digitalWrite(light, LOW);
      dacWrite(light, 0);
       Serial.println("# Light is OFF");
       Blynk.virtualWrite(V10, 0);
    }
}


BLYNK_WRITE(V12) // Executes when the value of virtual pin 5 changes
  {
    int alert_status = param.asInt();
    if (alert_status == 1) {
      setAlarm("w");
    }
}


BLYNK_WRITE(V13) // Executes when the value of virtual pin 5 changes
  {
    int alert_status = param.asInt();
    if (alert_status == 1) {
      setAlarm("d");
    }
}
```