JAYPEE INSTITUTE OF INFORMATION TECHNOLOGY
NOIDA

Smart Hard Helmet for Construction Workers

Submitted By:
Rachit Arora (21102028)
Shubh (21102029)

Submitted To:
Dr Ritu Raj

Minor-2 project report submitted in partial fulfilment of the requirement for the degree of

Bachelor of Technology in
Electronics and Communication Engineering

CERTIFICATE

This is to certify that Rachit Arora (21102028) and Shubh (21102029), students of Electronics and Communication Engineering, Jaypee Institute of Information Technology, Noida have completed the Minor-2 project on the topic "Smart Hard Helmet for Construction Workers" under the guidance of Dr Ritu Raj during the year 2024. This work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or diploma.

Signature of Supervisor …….

Name of Supervisor: Dr. Ritu Raj

Designation …….

Date: 06/05/2024

CANDIDATE'S DECLARATION

This is to certify that the work is being presented in the BTech Minor-2 Project report entitled "**Smart Hard Helmet For Construction Workers**", submitted by Rachit Arora (21102028) and Shubh (21102029), in partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology** in **Electronics & Communication Engineering** and submitted to the Department of Electronics & Communication Engineering of Jaypee Institute of Information Technology, Noida is an authentic record of our work carried out during a period from January 2024 to May 2024 under the supervision of **"DR RITU RAJ"**, ECE Department.

(Student Signature)
Rachit Arora

(Student Signature)
Shubh

Date: 06/05/2024

ABSTRACT

In our Minor Project-2, we tried to devise a PCB for a Hard helmet that construction workers use for safety purposes. In addition to saving their heads from severe injury, the "Smart" Hard helmet will be able to detect falls of the worker using MPU6050, will constantly monitor their heart rate and SpO2 percentage and a regular mail will be sent to the Thekedaar about the workers' heart rate and SpO2.This functionality has been realized using the MAX30100 sensor. The Helmet will also alert the workers and Thekedaar if the environment contains any flammable gas using an MQ-2 gas sensor.

Various methods have been tried and realized to prevent and minimize the percentage of fake or false Fall detections or Critical Heart Rate and SpO2 or Gas leakage.

For Heart Rate and SpO2, averaging of 10 continuous readings is done for accurate measurement and to avoid the influence of 0 values from the sensor, the ESP32 has been coded accordingly with various parameters and thresholds.

Similarly, a 3-level trigger system has been used to detect falls and to prevent false fall alarm making use of both accelerometer and gyroscope readings from MPU6050.

For a more vivid demonstration of the project, we have used different colour LEDs and a buzzer. If the system detects a fall, the green LED will turn on and simultaneously Buzzer will start buzzing until the system sends alert mail and restarts (around 5 secs). Similarly, Red LED is used for Gas Alerts, and for critical heart rate and SpO2 readings blue LED is used.

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to our supervisor Dr Ritu Raj for providing his invaluable guidance, comments, and suggestions throughout the project. He consistently motivated and guided us towards the completion of the project. We are highly indebted to sir for his constant supervision as well as for providing necessary information regarding the project. However, it would not have been possible without the kind support and encouragement of our parents and colleagues who have helped me out with their abilities in developing the project.

Signature …….

Rachit Arora, 21102028

Signature …….

Shubh, 21102029

Date: 6/05/2024

Table of Contents

Chapter 1: Smart Hard Helmet

1.1. INTRODUCTION

Internet The Smart Hard Helmet emerges as a beacon of innovation in the construction industry, amalgamating state-of-the-art technologies to usher in a new era of safety and well-being for workers. Within this revolutionary headgear, individual components such as the MAX30100 pulse oximeter, MPU6050 accelerometer and gyroscope, and MQ-2 gas sensor play pivotal roles.

The MAX30100 pulse oximeter, with its non-invasive monitoring capabilities, keeps a vigilant eye on the wearer's heart rate and oxygen saturation levels. This real-time health tracking is not merely a convenience but a life-saving feature, allowing for immediate response to any anomalies. Simultaneously, the MPU6050 accelerometer and gyroscope serve as guardians against physical risks, detecting sudden movements, falls, or collisions. This ensures swift intervention in the event of an accident, minimizing injuries and enhancing overall safety.

The integration of the MQ-2 gas sensor is a proactive measure against the often-unseen hazards in construction environments. By continuously monitoring for hazardous gases, the smart helmet offers an additional layer of protection, alerting workers to potential dangers and enabling swift evacuation or corrective actions.

The importance of the Smart Hard Helmet extends far beyond the realm of individual components. It addresses the pressing safety challenges inherent in the construction industry, where the risk of accidents and occupational hazards is omnipresent. By fostering a culture of continuous health monitoring, the helmet not only safeguards workers but also contributes to increased productivity by mitigating downtime caused by accidents or health issues.

In conclusion, the Smart Hard Helmet is not just a technological marvel; it is a guardian angel for construction workers. It blends the precision of individual sensors into a seamless safety net, transforming the construction site into a safer, healthier, and more productive environment. As this technology matures, the construction industry stands poised for a paradigm shift towards a future where every worker can confidently don their helmet, knowing that their safety is prioritized at every step.

1.2 COMPONENTS NEEDED

ESP-32: The heart of the system, responsible for connecting to the internet and processing data.

Accelerometer/Gyroscope Sensor (MPU60650): This sensor detects changes in acceleration and orientation, helping in identifying falls.

Heartrate and SpO2 Monitoring Sensor (MAX30100): This sensor detects heart rate and SpO2 by using IR and Red Light.

MQ-2 Gas Sensor: It is one of the most commonly used gas sensor to detect combustible gases like Butane.

Power Source: A reliable power supply to ensure continuous operation of the ESP-32 and sensors.

Internet Connection: Wi-Fi connectivity for the ESP-32 to transmit data to a cloud server.

LEDs: For a more vivid demonstration, different LEDs will tell about the state of the alert.

Buzzer: The Buzzer Module is used to alert other workers in case of emergency.

1.3 RELATED WORK AND MOTIVATION

The endeavour to research and develop Smart Hard Helmets for construction workers is rooted in a profound motivation to address the pressing safety challenges endemic to the construction industry. As a critical sector vital to societal progress, construction sites are marred by inherent occupational hazards that necessitate a transformative approach to safeguard the well-being of the workforce.

The integration of advanced sensors, including the MAX30100 pulse oximeter, MPU6050 accelerometer and gyroscope, and MQ-2 gas sensor, is driven by a desire for real-time health monitoring and proactive risk detection. The MAX30100 facilitates continuous health tracking, allowing for the prompt detection of irregularities in heart rate and oxygen saturation levels. Simultaneously, the MPU6050 serves as a sentinel against physical risks, detecting sudden movements and anomalies to prevent falls and collisions. The inclusion of the MQ-2 gas sensor is motivated by the hidden dangers of gas exposure in construction sites, aiming to proactively detect hazardous gases and provide early warnings.

The expected impact of this research is far-reaching, aiming to elevate worker safety standards, prioritize physical well-being, and contribute to increased productivity by minimizing downtime caused by accidents or health-related issues. Ultimately, the motivation driving the research on Smart Hard Helmets lies in the earnest pursuit of advancing worker safety in the construction industry, symbolizing a commitment to a safer future for those whose labor builds the foundation of our societies.

1.4 SYSTEM HARDWARE

The hardware configuration for an IoT-based fall detection system using the ESP32 is a pivotal element influencing the robustness and efficacy of the solution. The system orchestrates an

amalgamation of sensors, microcontrollers, and power sources to meticulously detect falls and transmit timely alerts. Here's an elucidation of the key components:

ESP32 Microcontroller: Serving as the central processing unit, the ESP32 is instrumental in collecting data from sensors, executing the fall detection algorithm, and facilitating communication with the cloud or external platforms. Its compact design, cost-effectiveness, and integrated Wi-Fi capabilities make it an ideal choice for IoT applications, ensuring seamless connectivity.

Accelerometer/Gyroscope Sensor: A fundamental component for discerning changes in acceleration and orientation. This sensor furnishes real-time data on user movements, empowering the fall detection algorithm to scrutinize patterns indicative of a fall. Optimal sensor quality with ample sensitivity is imperative for precise detection.

Heart Rate Monitor: Adding a heart rate monitor enhances the system's capabilities by providing additional health data for a comprehensive understanding of the user's well-being.

Gas Sensor: Integrating a gas sensor further fortifies the system against potential environmental hazards, providing early detection of harmful gases.

Power Source: A steadfast power supply is imperative for uninterrupted operation. Options encompass rechargeable batteries or direct power sources contingent on the deployment scenario. Battery efficiency assumes paramount importance to sustain prolonged system functionality without frequent replacements.

Protective Enclosure: This casing shields the components, ensuring durability and resilience against environmental conditions. The enclosure is meticulously designed to house the ESP32, sensors, and power source, incorporating adequate ventilation to avert overheating.

Wi-Fi Module: Given the ESP32's reliance on Wi-Fi for data transmission, a stable and reliable internet connection is paramount. The Wi-Fi module ensures seamless communication between the ESP32 and the cloud platform or remote monitoring system.

In summary, the comprehensive hardware configuration for an IoT-based fall detection system with the ESP32 encompasses the ESP32 microcontroller, accelerometer/gyroscope sensor, heart rate monitor, gas sensor, power source, protective enclosure, Wi-Fi module, and potentially additional sensors for augmented functionality. A meticulously designed and integrated hardware setup assures the system's reliability in accurately detecting falls and expediting timely responses for the safety and well-being of users.

1.5 SYSTEM OVERVIEW

The development of an IoT-based fall detection system utilizing the ESP32 involves a systematic approach that encompasses hardware integration, sensor data processing, algorithm implementation, and cloud connectivity. This overview outlines the key steps and components involved in creating a robust and effective fall detection solution.

1. Hardware Configuration:

- The foundation of the system lies in its hardware configuration. The central processing unit is the ESP32, chosen for its affordability, compact design, and built-in Wi-Fi capabilities. Paired with an accelerometer, gyroscope sensor, heart rate monitor, SpO2 sensor, and smoke sensor, it forms the core sensor module responsible for capturing real-time data on the user's movements, health metrics, and environmental conditions. Additional components include a reliable power source, an enclosure for protection, and, if necessary, supplementary sensors for a more comprehensive health monitoring system.

2. Sensor Data Acquisition:

- Accelerometer and Gyroscope Sensors: These play a pivotal role in capturing the user's movements, providing data on acceleration and orientation for the fall detection algorithm.

- Heart Rate Monitor and SpO2 Sensor: These sensors contribute to health monitoring, providing real-time data on the user's heart rate and blood oxygen saturation levels.

- Smoke Sensor: This environmental sensor detects the presence of smoke, enhancing the system's safety features.

3. Fall Detection Algorithm:

- The algorithm is the intelligence behind the system, responsible for analyzing sensor data and identifying patterns associated with falls. This involves setting thresholds for acceleration, understanding impact forces, and recognizing specific orientation changes. The algorithm must be finely tuned to distinguish between normal activities and genuine falls, minimizing false positives and negatives. Iterative testing and refinement are essential to enhance the algorithm's accuracy.

4. ESP32 Programming:

- The ESP32 needs to be programmed to execute the fall detection algorithm, process sensor data, and initiate an alert mechanism when a fall is detected. Programming is typically done using platforms like the Arduino IDE, leveraging the simplicity of the ESP32's programming language. The code must ensure efficient data processing, low power consumption, and seamless communication with the chosen cloud platform.

5. User Interface (Optional):

- For a more user-friendly experience, a web or mobile interface can be developed. This interface can display real-time fall alerts, historical data, health metrics, and additional

information from the environmental sensors. It serves as a communication bridge between the IoT system and caregivers, providing valuable insights into the user's well-being.

In summary, the IoT-based fall detection system using the ESP32 follows a comprehensive approach, integrating hardware components, sensor data processing, a sophisticated fall detection algorithm, ESP32 programming and, optionally, a user interface. This holistic system aims to enhance the safety and well-being of individuals by providing timely alerts, health insights, and environmental monitoring through continuous analysis.

Chapter 2: ESP 32

2.1 EXPLANATION OF THE ESP 32 PLATFORM

The ESP32 microcontroller serves as the core of the ESP-32 platform, an open-source Internet of Things (IoT) solution developed around the ESP8266 Wi-Fi module. Created by Espressif Systems, the ESP-32 platform has gained widespread popularity for its affordability, versatility, and user-friendly nature. This overview will explore the fundamental components and features that define the ESP32-based ESP-32 platform.

1. ESP32 Microcontroller:

At the heart of the ESP-32 platform is the ESP32 microcontroller. This powerful chip features a 32-bit RISC CPU, providing substantial computational capability for a variety of IoT applications. Notably, the ESP32 is recognized for its integrated Wi-Fi functionality, facilitating seamless communication with other devices and networks.

2. GPIO (General Purpose Input/Output) Pins:

The ESP-32 board is equipped with a set of GPIO pins serving as interfaces for both digital and analog signals. These pins enable developers to connect the ESP-32 to various sensors, actuators, and electronic components, expanding its functionality beyond Wi-Fi communication.

3. UART, SPI, I2C Interfaces:

To facilitate communication with peripherals, the ESP-32 platform features UART (Universal Asynchronous Receiver-Transmitter), SPI (Serial Peripheral Interface), and I2C (Inter-Integrated Circuit) interfaces. These standard communication protocols enhance the platform's compatibility with a wide array of sensors and devices, enabling the creation of diverse IoT projects.

4. Wi-Fi Connectivity:

A standout feature of the ESP-32 platform is its embedded Wi-Fi module. This capability empowers the ESP-32 to connect to local Wi-Fi networks, providing access to the internet and facilitating communication with other IoT devices. Wi-Fi integration makes ESP-32 an ideal choice for projects requiring remote monitoring, data exchange, and internet control.

5. Open-Source Platform:

ESP-32 is an open-source platform, that providing freely available hardware design and software code for modification and distribution. This fosters a collaborative development environment, encouraging a community of developers to contribute enhancements, bug fixes, and new features.

In conclusion, the ESP-32 platform seamlessly integrates the ESP32 microcontroller with Wi-Fi connectivity, GPIO pins, and standard communication interfaces. Its open-source nature and support for the Lua scripting language make it a versatile and accessible choice for IoT enthusiasts and developers. Whether used for prototyping or deploying IoT solutions, ESP-32 remains a preferred platform due to its blend of features, affordability, and strong community support.

2.2 SELECTION AND INTEGRATION OF ACCELEROMETER

The selection and integration of the accelerometer in an IoT-based fall detection system using ESP-32 are critical components that directly influence the accuracy and reliability of the solution. Accelerometers are essential for capturing changes in acceleration and orientation, providing real-time data that is crucial for the fall detection algorithm.

1. Selection Criteria:

When choosing an accelerometer, several factors must be considered to ensure its suitability for fall detection applications:

Sensitivity: The accelerometer must be sensitive enough to detect subtle changes in acceleration that may indicate a fall. Higher sensitivity allows for a more accurate analysis of the user's movements.

Range: A suitable range is required to accommodate both the normal activities of the user and the impact forces associated with falls. A wide range ensures that the accelerometer can capture a broad spectrum of movements.

Size and Form Factor: For wearable applications, the size and form factor of the accelerometer should be compact and lightweight to ensure user comfort. This is particularly important for fall detection systems that may be integrated into clothing or accessories.

2. Integration with ESP-32:

Once an appropriate accelerometer is selected, the integration with ESP-32 involves connecting the sensor to the microcontroller and programming it to read and process the accelerometer data. The following steps outline the integration process:

Wiring: Connect the accelerometer to the appropriate pins on the ESP-32. Most accelerometers communicate via I2C or SPI protocols, and the wiring configuration should adhere to the specifications provided by the accelerometer's datasheet.

Programming: Write code for the ESP-32 to initialize and read data from the accelerometer. This involves configuring the communication protocol, setting up data registers, and implementing a loop to continuously read sensor data.

Data Processing: Process the raw accelerometer data to extract meaningful information. In the context of fall detection, this may involve applying filtering techniques, analyzing acceleration patterns, and identifying specific signatures associated with falls.

Integration with Fall Detection Algorithm: Integrate the accelerometer data processing into the fall detection algorithm. The algorithm should interpret the acceleration patterns and make informed decisions based on predefined criteria to distinguish falls from normal activities.

Power Optimization: Implement power-saving measures to optimize the energy efficiency of the system. This may include adjusting the accelerometer's sampling rate based on activity levels or incorporating sleep modes to conserve power during periods of inactivity.

3. Calibration and Testing:

After integration, it's crucial to calibrate the system to account for variations in user behavior and sensor characteristics. Testing involves validating the system's performance under different scenarios, including simulated falls and daily activities. Iterative testing and refinement are essential to enhance the system's accuracy and reliability.

In conclusion, the selection and integration of the accelerometer in an IoT-based fall detection system using ESP-32 require careful consideration of sensitivity, range, sampling rate, and form factor. The integration process involves wiring, programming, data processing, and optimization for power efficiency. Calibration and thorough testing are essential to ensure the system's accuracy in detecting falls and minimizing false positives or negatives, contributing to the overall effectiveness of the fall detection solution.

Chapter 3: Fall Detection using MPU6050

3.1 DEVELOPMENT OF FALL DETECTION ALGORITHM

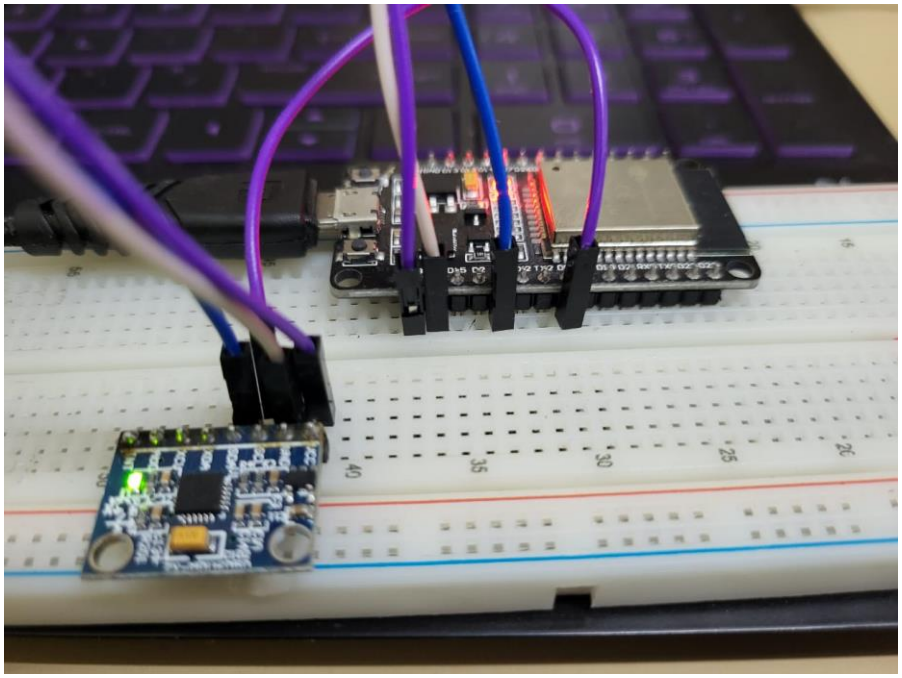

Fig. MPU6050 connected with ESP-32

The development of a robust fall detection algorithm is a pivotal aspect of creating an effective IoT-based fall detection system using ESP-32. This algorithm serves as the intelligence behind the system, analysing data from the accelerometer to identify patterns indicative of falls while minimizing false positives and negatives. Here is an overview of the key steps involved in developing the fall detection algorithm:

1. Data Preprocessing:

The first step in the algorithm development process is preprocessing the raw data obtained from the accelerometer. This involves filtering and smoothing techniques to remove noise and ensure that the algorithm works with clean and accurate data. Techniques such as low-pass filters can be applied to attenuate high-frequency noise, providing a more reliable input for subsequent analysis.

2. Feature Extraction:

Feature extraction involves identifying relevant characteristics or features from the pre-processed data that are indicative of a fall event. Accelerometer data typically consists of three axes (X, Y, Z), and features such as maximum acceleration, jerk, or changes in orientation can be extracted. These features act as input parameters for the fall detection algorithm.

3. Threshold Setting:

Setting appropriate thresholds for the extracted features is crucial for distinguishing between normal activities and fall events. Thresholds are defined based on the characteristics of the features and are determined through empirical analysis and testing. Adjusting these thresholds allows for fine-tuning the algorithm to achieve a balance between sensitivity and specificity.

4. Algorithm Logic:

The core logic of the fall detection algorithm is designed to analyse the feature values in real-time and trigger an alert when a fall is detected. This logic should account for variations in individual behaviour and the dynamic nature of daily activities. Machine learning techniques can be incorporated for adaptive threshold setting and continuous improvement of the algorithm's performance over time.

5. Time-Series Analysis:

Considering the temporal aspect of accelerometer data is essential. Falls often exhibit distinctive patterns over time, and analysing the sequential changes in acceleration can enhance the accuracy of detection. Time-series analysis techniques, such as dynamic time warping or pattern recognition algorithms, can be applied to capture the temporal characteristics of fall events.

6. Validation and Calibration:

The fall detection algorithm should undergo rigorous validation and calibration to ensure its reliability across diverse scenarios. Testing involves using real-world data, including simulated

falls and various daily activities. Calibration is an iterative process, refining the algorithm based on feedback from testing to improve its accuracy and reduce false positives or negatives.

7. Integration with ESP-32:

The final step involves integrating the developed fall detection algorithm with the ESP-32's programming. The algorithm is embedded in the code running on the ESP-32, enabling it to process accelerometer data in real time. The code should include logic for triggering alerts, communicating with the cloud, and managing the overall system functionality.

In summary, the development of a fall detection algorithm for an IoT-based system using ESP-32 involves data preprocessing, feature extraction, threshold setting, algorithm logic design, time-series analysis, and rigorous validation. The goal is to create an intelligent algorithm that can accurately and reliably identify falls while minimizing false alarms, thereby enhancing the safety and well-being of individuals in need of fall detection assistance.

3.2 In-depth explanation of fall detection algorithm

The fall detection algorithm in an IoT-based system using ESP-32 is a critical component designed to analyse accelerometer data and accurately identify potential fall events while minimizing false positives. The algorithm goes through several stages to ensure robustness and reliability in detecting falls.

1. Data Preprocessing:

Raw data from the accelerometer, capturing acceleration and orientation, undergoes preprocessing to eliminate noise and ensure the algorithm operates with clean and accurate data. Filtering techniques, such as low-pass filters, are applied to attenuate high-frequency noise, providing a more reliable foundation for subsequent analysis.

2. Feature Extraction:

The algorithm extracts relevant features from the pre-processed accelerometer data. These features serve as key indicators of fall events and include parameters like maximum acceleration, jerk, and changes in orientation. Feature extraction is crucial for transforming raw sensor data into meaningful variables that the algorithm can analyse.

3. Threshold Setting:

Setting appropriate thresholds for the extracted features is a vital step in distinguishing normal activities from fall events. Thresholds are determined through empirical analysis and testing, considering the characteristics of the features. Adjusting these thresholds allows for fine-tuning the algorithm, striking a balance between sensitivity (correctly identifying falls) and specificity (avoiding false positives).

4. Pattern Recognition and Time-Series Analysis:

The algorithm utilizes pattern recognition techniques and time-series analysis to identify distinctive patterns associated with falls. Falls often exhibit specific temporal characteristics, such as rapid changes in acceleration followed by impact forces. Analyzing these patterns enhances the algorithm's ability to discriminate between falls and other activities.

5. Machine Learning Integration (Optional):

In more advanced implementations, machine learning techniques can be integrated to enhance the algorithm's adaptability and predictive capabilities. Supervised learning models can be trained on diverse datasets, allowing the algorithm to continuously learn and improve its performance over time. Machine learning adds a layer of adaptability to accommodate variations in individual behaviour.

6. Validation and Calibration:

Rigorous validation is conducted using real-world data, including simulated falls and various daily activities. The algorithm's performance is assessed based on metrics such as sensitivity, specificity, and accuracy. Calibration is an iterative process, refining the algorithm based on

feedback from testing to improve its reliability and reduce the likelihood of false positives or negatives.

7. Alert Triggering Mechanism:

When the algorithm identifies a potential fall event based on the established criteria, it triggers an alert mechanism. This mechanism can take the form of notifications sent to caregivers or emergency services, providing essential details such as the timestamp, location (if available), and user ID.

8. Post-Fall Actions and User Feedback:

Post-fall actions may involve the initiation of emergency protocols or notifying designated contacts. In some implementations, a feedback loop allows users or caregivers to confirm whether the fall detection was accurate. This feedback contributes to ongoing improvements in the algorithm.

9. Integration with Cloud Server and Mobile App:

The fall detection algorithm is integrated into the ESP-32's programming, enabling seamless communication with the cloud server and mobile app. The algorithm's results, including fall alerts and relevant data, are transmitted to the cloud for storage, analysis, and remote monitoring through a user-friendly mobile app.

In summary, the fall detection algorithm for an IoT-based system using ESP-32 is a multifaceted process involving data preprocessing, feature extraction, threshold setting, pattern recognition, optional machine learning integration, rigorous validation, and seamless integration with cloud services and user interfaces. This comprehensive approach ensures that the algorithm is not only accurate in detecting falls but also adaptable to varying scenarios and continuously improving over time.

3.3 IN-DEPTH EXPLANATION OF FALL DETECTION ESP-32 CODE:

1. Include Libraries and declare Variables:

```
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
#include "MAX30100_PulseOximeter.h"
#define REPORTING_PERIOD_MS 50
const int MPU_addr = 0x68;
Adafruit_MPU6050 mpu;
PulseOximeter pox;
uint32_t tsLastReport = 0;
int16_t AcX, AcY, AcZ, Tmp, GyX, GyY, GyZ;
float ax = 0, ay = 0, az = 0, gx = 0, gy = 0, gz = 0;
boolean fall = false;
boolean trigger1 = false;
boolean trigger2 = false;
boolean trigger3 = false;
byte trigger1count = 0;
byte trigger2count = 0;
byte trigger3count = 0;
int angleChange = 0;
```

These lines include necessary libraries for working with the MPU6050 sensor and I2C communication. Also, various variables have been declared to store the raw values from sensor and triggers to be used in detection.

2. Fetching raw values from MPU pins:

```
void mpu_read() {
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x3B);
  Wire.endTransmission(false);
  Wire.requestFrom(MPU_addr, 14, true);
  AcX = Wire.read() << 8 | Wire.read();
  AcY = Wire.read() << 8 | Wire.read();
  AcZ = Wire.read() << 8 | Wire.read();
  Tmp = Wire.read() << 8 | Wire.read();
  GyX = Wire.read() << 8 | Wire.read();
  GyY = Wire.read() << 8 | Wire.read();
  GyZ = Wire.read() << 8 | Wire.read();
}
```

Using the Wire library and its functions, we fetch raw values of AcX(Acceleration around X axis), AcY, AcZ, GyX(Gyroscope value around X axis), GyY and GyZ.

3. Initialize MPU6050:

```
Serial.println("Initializing sensors...");
if (!mpu.begin()) {
  Serial.println("Failed to find MPU6050 chip");
  while (1) {
    delay(10);
  }
}
Serial.println("MPU6050 Found!");
mpu.setHighPassFilter(MPU6050_HIGHPASS_0_63_HZ);
mpu.setMotionDetectionThreshold(1);
mpu.setMotionDetectionDuration(10);
mpu.setInterruptPinLatch(true);
mpu.setInterruptPinPolarity(true);
mpu.setMotionInterrupt(true);
```

These lines of code help in initializing the mpu accordingly.

4. Fall Detection Logic:

```
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;
float Raw_Amp = pow(pow(ax, 2) + pow(ay, 2) + pow(az, 2), 0.5);
int Amp = Raw_Amp * 10;
Serial.println(Amp);
if (Amp <= 2 && trigger2 == false) {
  trigger1 = true;
  Serial.println("TRIGGER 1 ACTIVATED");
}
if (trigger1 == true) {
  trigger1count++;
  if (Amp >= 12) {
    trigger2 = true;
    Serial.println("TRIGGER 2 ACTIVATED");
    trigger1 = false;
    trigger1count = 0;
  }
}
if (trigger2 == true) {
  trigger2count++;
  angleChange = pow(pow(gx, 2) + pow(gy, 2) + pow(gz, 2), 0.5);
  Serial.println(angleChange);
  if (angleChange >= 30 && angleChange <= 400) {
    trigger3 = true;
    trigger2 = false;
    trigger2count = 0;
```

```
      Serial.println(angleChange);
      Serial.println("TRIGGER 3 ACTIVATED");
    }
  }
  if (trigger3 == true) {
    trigger3count++;
    if (trigger3count >= 10) {
      Serial.println("Trigger3 count = ");
      Serial.println(trigger3count);
      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(angleChange);
      } else {
        trigger3 = false;
        trigger3count = 0;
        Serial.println("TRIGGER 3 DEACTIVATED");
      }
    }
  }
  if (fall == true) {
    digitalWrite(LED_PIN1, HIGH);
    digitalWrite(BUZZER_PIN, HIGH);
    Serial.println("FALL DETECTED");
    Heart(true);
    sendEmail(avg_Hr, avg_SpO2, true);
    fall = false;
    delay(5000);
    resetFunc();
  }
  if (trigger2count >= 6) {
    trigger2 = false;
    trigger2count = 0;
    Serial.println("TRIGGER 2 DECACTIVATED");
  }
  if (trigger1count >= 6) {
    trigger1 = false;
    trigger1count = 0;
    Serial.println("TRIGGER 1 DEACTIVATED");
  }
  Heart(false);
  int sensorValue = analogRead(MQ2_AOUT);
  int digitalValue = digitalRead(MQ2_DOUT);
  Gas_mail(sensorValue, digitalValue);
  delay(100);
}
```

The system uses an MPU (Motion Processing Unit) to read acceleration and gyroscope data, which is then used to detect a fall. Here's a breakdown of the code:

Data Acquisition: The mpu_read(); function reads data from the MPU. The acceleration (ax, ay, az) and gyroscope (gx, gy, gz) values are calculated by subtracting a bias from the raw data and scaling it according to the sensitivity of the sensor.

Amplitude Calculation: The Raw_Amp is the magnitude of the acceleration vector, calculated using the Pythagorean theorem. This value is then scaled by a factor of 10 to get Amp.

Trigger 1 Activation: If Amp is less than or equal to 2 and trigger2 is false, trigger1 is activated. This could represent a condition where the person is still or not moving much.

Trigger 2 Activation: If trigger1 is true and Amp is greater than or equal to 12, trigger2 is activated and trigger1 is deactivated. This could represent a sudden movement, such as a fall.

Trigger 3 Activation: If trigger2 is true and the change in angle (calculated as the magnitude of the gyroscope vector) is between 30 and 400, trigger3 is activated and trigger2 is deactivated.

Fall Detection: If trigger3 is true and the change in angle is between 0 and 10 after a certain count (trigger3count), a fall is detected. The LED and buzzer are activated, an email is sent with heart rate and SpO2 data, and the system is reset after a delay.

Trigger Deactivation: If the count for trigger1 or trigger2 exceeds a certain limit (6 in this case), the respective trigger is deactivated. This could be a timeout condition to prevent false positives.

This code represents a state machine with different triggers representing different states of movement. The system transitions between these states based on the sensor data, and a specific sequence of states is used to detect a fall. This approach can help reduce false positives and improve the accuracy of fall detection.

Chapter 4: Heart Rate Monitoring and SpO2 Detection

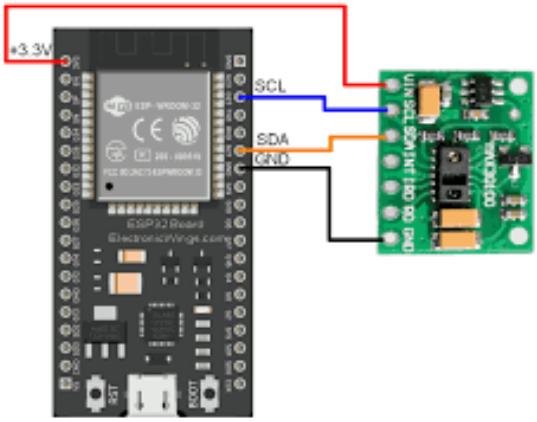

Fig. MAX30100 with ESP-32 DevKit V1

4.1 HEART RATE MONITORING ALGORITHM USING MAX30100 AND ESP32:

Here are the key points about the MAX30100 pulse oximeter and heart rate sensor:

1. Hardware Overview: The MAX30100 module features two LEDs (RED and IR) and a sensitive photodetector. By shining a single LED at a time and detecting the amount of light shining back at the detector, it can measure blood oxygen level and heart rate.

2. Power Requirement: The MAX30100 chip requires two different supply voltages: 1.8V for the IC and 3.3V for the RED and IR LEDs. The module comes with 3.3V and 1.8V regulators, allowing you to connect the module to any microcontroller with 5V, 3.3V, or even 1.8V level I/O.

3. Low Power Consumption: The MAX30100 consumes less than 600µA during measurement and only 0.7µA in standby mode. This allows implementation in battery-powered devices such as handsets, wearables, or smartwatches.

4. On-Chip Temperature Sensor: The MAX30100 has an on-chip temperature sensor that can be used to compensate for changes in the environment and to calibrate the measurements. It measures the 'die temperature' in the range of -40˚C to +85˚C with an accuracy of ±1˚C.

5. I2C Interface: The module uses a simple two-wire I2C interface for communication with the microcontroller. It has a fixed I2C address: 0xAEHEX (for write operation) and 0xAFHEX (for read operation).

6. FIFO Buffer: The MAX30100 embeds a FIFO buffer for storing data samples. The FIFO has a 16-sample memory bank, which means it can hold up to 16 SpO2 and heart rate samples. The FIFO buffer can offload the microcontroller from reading each new data sample from the sensor, thereby saving system power.

7. Interrupts: The MAX30100 can be programmed to generate an interrupt, allowing the host microcontroller to perform other tasks while the data is collected by the sensor. The interrupt can be enabled for 5 different sources: Power Ready, SpO2 Data Ready, Heart Rate Data Ready, Temperature Ready, and FIFO Almost Full.

8. Technical Specifications: The power supply ranges from 3.3V to 5.5V. The current draw is ~600μA during measurements and ~0.7μA during standby mode. The Red LED Wavelength is 660nm and the IR LED Wavelength is 880nm. The temperature range is -40˚C to +85˚C with an accuracy of ±1˚C.

9. MAX30100 vs. MAX30102: The MAX30102 sensor has a 32-sample memory bank, 18-bit ADC resolution, and a narrower LED pulse width than the MAX30100, resulting in lower power consumption.

10. Working Principle: The MAX30100 works by shining both lights onto the finger or earlobe and measuring the amount of reflected light using a photodetector. This method of pulse detection

through light is called Photoplethysmogram. The working of MAX30100 can be divided into two parts: Heart Rate Measurement and Pulse Oximetry (measuring the oxygen level of the blood).

11. Module Pinout: The MAX30100 module brings out connections for power (VIN), I2C clock (SCL), I2C data (SDA), interrupt (INT), IR LED driver (IRD), Red LED driver (RD), and ground (GND).

12. Troubleshooting: If you're having trouble seeing a heartbeat, try adjusting the pressure applied to the sensor, or try the sensor on different parts of your body that have capillary tissue.

4.2 IN-DEPTH EXPLANATION OF HEARTRATE AND SPO2 MONITORING USING ESP-32 CODE:

1. Include Libraries:

```
#include "MAX30100_PulseOximeter.h"
```

These lines include the necessary libraries for I2C communication (Wire) and the MAX30100 Pulse Oximeter.

2. Define Constants:

```
int avg = 0;
float avg_Hr = 0;
float avg_SpO2 = 0;
void onBeatDetected() {
  Serial.println("Beat!");
}
```

These lines define variables for average heart rate and average SpO2 of 10 consecutive values.

3. Initialize Pulse Oximeter:

```
PulseOximeter pox;
```

This line creates an instance of the PulseOximeter class.

4. Callback Function for Beat Detection:

```
void onBeatDetected()
{
    Serial.println("Beat!");
}
```

This function is a callback that gets triggered when a heartbeat is detected. In this case, it prints "Beat!" to the serial monitor.

5. Setup Function:

```
if (!pox.begin()) {
  Serial.println("Failed to find MAX30100 chip");
  for (;;)
    ;
}
Serial.println("MAX30100 Found!");
pox.setOnBeatDetectedCallback(onBeatDetected);
Serial.println("");
Wire.begin();
```

- The setup function initializes I2C communication, serial communication, and the Pulse Oximeter.

- If initialization fails, it prints "Failed to find MAX30100 chip" and enters an infinite loop. Otherwise, it prints "MAX30100 Found!" and sets the beat detection callback.

5. Loop Function:

```
float Hr[10] = { 0 };
float SpO2_Arr[10] = { 0 };
void Heart(bool check_fall) {
  if (millis() - tsLastReport > 5000) {
    pox.update();
    float Heart_rate = pox.getHeartRate();
    float SpO2 = pox.getSpO2();
    Serial.print("Heart rate:");
    Serial.print(Heart_rate);
    Serial.print("bpm / SpO2:");
    Serial.print(SpO2);
    Serial.println("%");
    Serial.print(avg);
    Serial.println("int");
    Hr[avg] = Heart_rate;
    SpO2_Arr[avg] = SpO2;
    avg++;
    if (avg == 10) {
      int n = foundzeroesinSpO2( Hr,SpO2_Arr);
      int n1 = CriticalSpO2(SpO2_Arr);
      for (int i = 0; i < 10; i++) {
        Serial.print(Hr[i]);
        Serial.print(" ");
        Serial.print(SpO2_Arr[i]);
        Serial.println("  ");
      }
      Serial.print("Avg Heart rate:");
      Serial.print(avg_Hr);
      Serial.print("bpm / Avg SpO2:");
      Serial.print(avg_SpO2);
      Serial.print("%  /  Error Value:");
      Serial.println(n);
```

```
      Serial.println(n);
      if ( avg_SpO2 < 90 ) {
        digitalWrite(LED_PIN3, HIGH);
        digitalWrite(BUZZER_PIN, HIGH);
        sendEmail_SpO2(avg_SpO2);
        resetFunc();
      }
      if (avg_Hr > 170 || avg_Hr < 50) {
        digitalWrite(LED_PIN3, HIGH);
        digitalWrite(BUZZER_PIN, HIGH);
        sendEmail(avg_Hr, avg_SpO2, false);
        resetFunc();
      }
      if (avg_SpO2 > 100 || avg_Hr < 80) {
        digitalWrite(LED_PIN3, HIGH);
        digitalWrite(BUZZER_PIN, HIGH);
        sendEmail(avg_Hr, avg_SpO2, false);
        resetFunc();
      } else {
        sendEmail(avg_Hr, avg_SpO2, false);
        resetFunc();
      }
      avg_Hr = 0;
      avg_SpO2 = 0;
    }
    if (check_fall == true) {
      avg_Hr /= avg;
      avg_SpO2 /= avg;
    }
    tsLastReport = millis();
  }
}
```

The code can be explained in the following lines:

Initialization: Two arrays, Hr and SpO2_Arr, each of size 10, are initialized to store the heart rate and SpO2 values respectively.

Heart Function: The Heart(bool check_fall) function is defined to update the heart rate and SpO2 values and perform certain actions based on these values.

Time Check: The function first checks if 5000 milliseconds (5 seconds) have passed since the last report. If not, it exits the function.

Sensor Update: If 5 seconds have passed, the pulse oximeter sensor (pox) is updated.

Heart Rate and SpO2 Measurement: The heart rate and SpO2 values are obtained from the sensor and printed to the serial monitor.

Array Update: The current heart rate and SpO2 values are stored in the Hr and SpO2_Arr arrays respectively.

Average Calculation: Once 10 values have been stored in the arrays, the average heart rate and SpO2 values are calculated.

Condition Checks and Actions: If the average SpO2 is less than 90, or the average heart rate is greater than 170 or less than 50, or the SpO2 is greater than 100 or the heart rate is less than 80, certain actions are performed. These actions include turning on an LED and a buzzer, sending an email, and resetting the function.

Resetting Averages: After performing the necessary actions, the average heart rate and SpO2 values are reset to 0.

Fall Check: If the check_fall parameter is true, the average heart rate and SpO2 values are divided by the number of measurements (avg).

Time Update: The time of the last report is updated to the current time.

In conclusion, this code is a crucial part of a health monitoring system, providing real-time updates on a person's heart rate and SpO2 levels. It's a great example of how technology can be used to enhance healthcare and potentially save lives.

7. Avoiding faulty and zero values:

```
int foundzeroesinSpO2(float Heart[],float SpO2[]) {
  int count = 0;
  for (int i = 0; i < 10; i++) {
    if (SpO2[i] < 90.0 || Heart[i]>(220-22)) {
      count++;
    }
    else{
      avg_Hr += Heart[i];
      avg_SpO2 += SpO2[i];
    }
  }
    avg_Hr /= (10 - count);
    avg_SpO2 /= (10 - count);
    return count;
}
int CriticalSpO2(float SpO2[]){
  int count1 = 0;
  for (int i = 0; i < 10; i++) {
    if (SpO2[i] < 95 || SpO2[i] > 80) {
      count1++;
    }
  }
  if (count1 > 6) {
    return 1;
  }
  else
  return 0;
}
```

The function foundzeroesinSpO2 takes heart rate and SpO2 values as input, counts the number of faulty readings, and calculates the average of valid readings. Faulty readings are defined as SpO2 less than 90.0 or heart rate outside the range of 198-242. The function CriticalSpO2 checks if more than six SpO2 values are outside the range of 80-95. If so, it returns 1, indicating a critical condition; otherwise, it returns 0. These functions help ensure data reliability in a health monitoring system by filtering out erroneous readings.

Chapter 5: MQ-2 Gas Detection Algorithm

5.1 COMBUSTIBLE GAS DETECTION ALGORITHM USING MQ2 AND ESP32:

The MQ-2 is a gas sensor module that can be used to detect various gases such as methane, propane, butane, alcohol, smoke, and carbon monoxide. When combined with an ESP32 microcontroller, it becomes a part of a gas detection system. Let's break down the analysis:

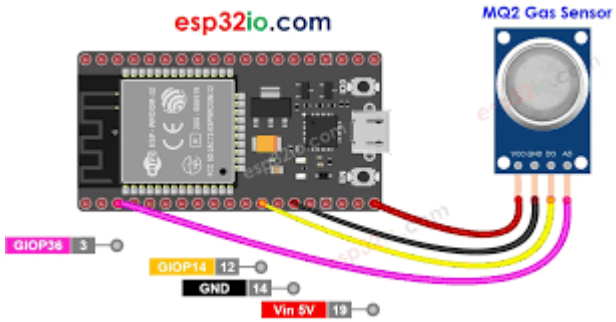

Fig. ESP-32 Connections with MQ-2 Gas Sensor

Properties of MQ-2 Gas Sensor:

1. Gas Detection:

- The MQ-2 is designed to detect a variety of gases in the surrounding environment, making it versatile for different applications.

2. Sensitivity to Multiple Gases:

- The sensor exhibits sensitivity to gases like LPG, i-butane, methane, smoke, alcohol, and carbon monoxide.

3. Analog Output:

- The sensor provides an analog output voltage proportional to the concentration of the detected gas. This analog signal can be read by the ESP32's analog-to-digital converter (ADC) for further processing.

4. Digital Output (Optional):

- Some versions of the MQ-2 sensor come with a digital output pin that provides a high or low signal based on a predefined threshold level. This can simplify the interface with digital input pins on the ESP32.

5. Preheat Time:

- The MQ-2 sensor requires a preheat time before it can provide stable and accurate readings. This preheat time is usually a few minutes.

6. Operating Voltage:

- The sensor typically operates at a voltage between 5V and 12V. Ensure that the ESP32 and sensor are powered within their specified voltage ranges.

5.2 ANALYSIS OF USAGE WITH ESP32:

1. Connection with ESP32:

- Connect the analog output of the MQ-2 sensor to one of the analog pins on the ESP32 (using a voltage divider if needed).

- If the sensor has a digital output, connect it to a digital pin on the ESP32.

2. Analog Readings:

- Read analog values from the sensor using the ESP32's ADC. These values represent the concentration of the detected gas.

3. Calibration:

- Calibrate the sensor for specific gases and concentrations based on the datasheet. Calibration ensures accurate readings.

4. Threshold Setting:

- Set threshold levels for each gas to trigger alerts or actions when the concentration exceeds a certain value.

5. Data Logging (Optional):

- Implement data logging on the ESP32 to record gas concentration levels over time for analysis or monitoring.

6. Alert Mechanism:

- Implement an alert mechanism, such as sending notifications or activating a buzzer, when the gas concentration surpasses the set thresholds.

7. Power Management:

- Consider power management strategies to optimize the usage of the ESP32 and prolong the system's battery life if applicable.

8. Environmental Conditions:

- Be aware of the environmental conditions that may affect sensor accuracy, such as temperature and humidity.

By integrating the MQ-2 sensor with an ESP32, you can create a gas detection system suitable for applications like gas leakage detection, smoke detection, and air quality monitoring. Ensure

that you refer to the datasheets of both the MQ-2 sensor and the ESP32 for accurate specifications and usage guidelines.

5.3 IN DEPTH EXPLANATION OF GAS SENSOR CODE:

1.Variable Declaration

```
int MQ2_AOUT;  //Initialization for MQ2
int MQ2_DOUT;
int threshold = 2000;
float Avg_Gas = 0;
int Avg_Gas_Count = 0;
```

Pins are declared for analog and digital output and variables are also declared to be used for threshold and averaging.

2. Pins Declaration

```
MQ2_AOUT = A0;
MQ2_DOUT = 13;
```

Analog output is set on A0 or pin 34 while digital output is set on pin 13.

3. Inside Loop function and Gas_mail function:

```
  Gas_mail(sensorValue, digitalValue);
  delay(100);
}
void Gas_mail(int sensorValue, int digitalValue) {
  Avg_Gas_Count++;
  Avg_Gas += sensorValue;
  Serial.print("Gas Value: ");
  Serial.println(sensorValue);
  if (Avg_Gas_Count == 10) {
    Avg_Gas /= 10;
    if (Avg_Gas > threshold) {
      digitalWrite(LED_PIN2, HIGH);
      digitalWrite(BUZZER_PIN, HIGH);
      sendEmail_Gas(Avg_Gas, true);
      resetFunc();
    } else {
      Serial.println("No gas detected.");
    }
    Avg_Gas_Count = 0;
  }
```

Gas_mail( ) function is called inside loop and average of the 10 values of previous analog values are done. If the average came out to be more than threshold then, Buzzer and LED2 will turn on and mail will be sent.

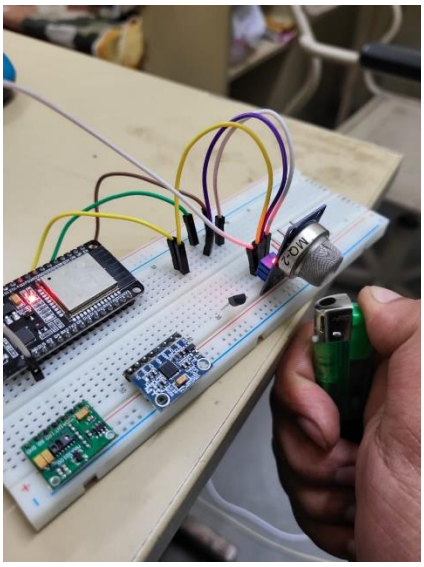

Fig. Using Lighter in front of MQ-2

Chapter 6: SMTP Protocol and Email System

6.1 SMTP PROTOCOL

The Simple Mail Transfer Protocol (SMTP) is an internet standard for email transmission. It's a communication protocol that mail servers use to send and receive emails over the internet. When it comes to the ESP32, a microcontroller with built-in Wi-Fi capabilities, it can be programmed to send emails using the SMTP protocol. This is achieved by connecting the ESP32 to an SMTP server.

To facilitate this process, a library such as the ESP-Mail-Client library is typically used. This library provides the necessary functions that allow the ESP32 to send and receive emails with or without attachments via SMTP and IMAP servers.

In a typical setup, the ESP32 acts as an email client. It connects to the SMTP server using the server's address and port number. It then sends an email composed of a header and a body. The header contains information such as the sender's and recipient's email addresses, the subject of the email, and the time and date of sending. The body contains the actual content of the email.

The ESP32 can send various types of content in an email. This includes simple text, HTML messages, images, and even files. The files can be stored in the ESP32's filesystem or on a microSD card.

This capability of the ESP32 to send emails programmatically makes it a powerful tool in IoT applications. For instance, it can be used to send sensor readings, alerts, or log files to a user's email. This allows for remote monitoring and data logging, which can be particularly useful in home automation, industrial automation, and various other IoT applications.

6.2 CODE FOR SENDING MAIL

1.Setup Code

```
#if defined(ESP32)
#include <WiFi.h>
#elif defined(ESP8266)
#include <ESP8266WiFi.h>
#endif

#include <ESP_Mail_Client.h>

#define WIFI_SSID "Galaxy S21 FE 5G 22FE"
#define WIFI_PASSWORD "yaow3245"
#define SMTP_HOST "smtp.gmail.com"
#define SMTP_PORT 465
#define AUTHOR_EMAIL "worker.safety048@gmail.com"
#define AUTHOR_PASSWORD "zirk ahxm sjub plyd"
#define RECIPIENT_EMAIL "shubhkumar802@gmail.com"

SMTPSession smtp;
void smtpCallback(SMTP_Status status);
Session_Config config;

void setup() {
  Serial.begin(9600);
  Serial.println();
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  Serial.print("Connecting to Wi-Fi");
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    delay(300);
  }
  Serial.println();
  Serial.print("Connected with IP: ");
  Serial.println(WiFi.localIP());
  Serial.println();
  MailClient.networkReconnect(true);
  smtp.debug(1);
  smtp.callback(smtpCallback);
  config.server.host_name = SMTP_HOST;
  config.server.port = SMTP_PORT;
  config.login.email = AUTHOR_EMAIL;
  config.login.password = AUTHOR_PASSWORD;
  config.login.user_domain = "";
  config.time.ntp_server = F("pool.ntp.org,time.nist.gov");
  config.time.gmt_offset = 5 * 60 + 30;
  config.time.day_light_offset = 0;
  while (!Serial)
    delay(10);
```

This is the setup code for establishing a connection and giving Wi-Fi access to the esp32 Wi-Fi module.

2. Making specific email functions for specific components

```
void sendEmail_Gas(float sensorValue, bool isAlert = false) {
  SMTP_Message message;
  message.sender.name = F("ESP");
  message.sender.email = AUTHOR_EMAIL;
  message.subject = isAlert ? F("Gas Alert") : F("Gas Update");
  message.addRecipient(F("Shubh"), RECIPIENT_EMAIL);
  String textMsg;
  if (sensorValue == 0) {
    textMsg = "No Gas detected.";
  } else {
    textMsg = "Gas Value: " + String(sensorValue);
  }
  message.text.content = textMsg.c_str();
  message.text.charSet = "us-ascii";
  message.text.transfer_encoding = Content_Transfer_Encoding::enc_7bit;
  message.priority = esp_mail_smtp_priority::esp_mail_smtp_priority_low;
  message.response.notify = esp_mail_smtp_notify_success | esp_mail_smtp_notify_failure | esp_mail_smtp_notify_delay;
  if (!smtp.connect(&config)) {
    ESP_MAIL_PRINTF("Connection error, Status Code: %d, Error Code: %d, Reason: %s", smtp.statusCode(), smtp.errorCode(), smtp.errorReason().c_str());
    return;
  }
  if (!smtp.isLoggedIn()) {
    Serial.println("\nNot yet logged in.");
  } else {
    if (smtp.isAuthenticated())
      Serial.println("\nSuccessfully logged in.");
    else
      Serial.println("\nConnected with no Auth.");
  }
  if (!MailClient.sendMail(&smtp, &message))
    ESP_MAIL_PRINTF("Error, Status Code: %d, Error Code: %d, Reason: %s", smtp.statusCode(), smtp.errorCode(), smtp.errorReason().c_str());
`
```

```
void sendEmail(float heartRate, float spO2, bool isAlert) {
  SMTP_Message message;
  message.sender.name = F("ESP");
  message.sender.email = AUTHOR_EMAIL;
  if (isAlert == true) {
    message.subject = ("Fall Detection Alert");
  }
  if (isAlert == false) {
    message.subject = ("Regular Health Update");
  }
  message.addRecipient(F("Shubh"), RECIPIENT_EMAIL);
  String textMsg;
  if (heartRate == 0 || spO2 == 0) {
    textMsg = "No readings detected from MAX30100";
  }
  if (isAlert == true) {
    textMsg = "Fall Detected!!!  ";
  }
  if (isAlert == false) {
    textMsg = "Heart rate: " + String(heartRate) + "bpm / SpO2: " + String(spO2) + "%";
  }
  message.text.content = textMsg.c_str();
  message.text.charSet = "us-ascii";
  message.text.transfer_encoding = Content_Transfer_Encoding::enc_7bit;
```

```
void sendEmail_SpO2(float spO2) {
  SMTP_Message message;
  message.sender.name = F("ESP");
  message.sender.email = AUTHOR_EMAIL;

  message.subject = ("Critical SpO2 Alert");

  message.addRecipient(F("Shubh"), RECIPIENT_EMAIL);
  String textMsg;

  textMsg = " SpO2: " + String(spO2) + "%";

  message.text.content = textMsg.c_str();
  message.text.charSet = "us-ascii";
  message.text.transfer_encoding = Content_Transfer_Encoding::enc_7bit;
  message.priority = esp_mail_smtp_priority::esp_mail_smtp_priority_low;
  message.response.notify = esp_mail_smtp_notify_success | esp_mail_smtp_notify_failure | esp_mail_smtp_notify_delay;
  if (!smtp.connect(&config)) {
```

Different functions have been devised for Max, Mpu, and MQ-2 and these are called accordingly.

As understood by seeing the code only, sendEmail_Gas() will be called when the Gas value goes above the threshold. Similarly, sendEmail_SpO2() will be called when the SpO2 falls into the critical range of 95 to 80.

The sendEmail() function serves two purposes. First of all, it sends a regular mail of heart rate and SpO2 measurement every 1 min and if the Alert variable is true, it will serve its second purpose by sending the fall detection alert mail.

Chapter 7: Buzzer and LEDs

7.1 ANALYSIS OF USAGE WITH ESP-32

Additionally, for a good demonstration of all the functionalities the system is embedded with, buzzer and LEDs are used. For critical Heart-rate or SpO2 or we could say for MAX30100 critical values, the blue LED will turn on along with the Buzzer for approximately 5 secs. For MPU6050, a green LED will turn on along with the buzzer if a fall is detected. And, a Red LED will turn on if a flammable gas is detected.

7.2 IMPLEMENTATION IN CODE

1.Declaration of pins to be used

```
#include <Wire.h>
#define LED_PIN1 5
#define LED_PIN2 18
#define LED_PIN3 19
#define BUZZER_PIN 23
```

2.Setting all pins with their mode and putting them at LOW

```
pinMode(MQ2_DOUT, INPUT);
pinMode(LED_PIN1, OUTPUT);
pinMode(LED_PIN2, OUTPUT);
pinMode(LED_PIN3, OUTPUT);
pinMode(BUZZER_PIN, OUTPUT);
digitalWrite(LED_PIN1, LOW);
digitalWrite(LED_PIN2, LOW);
digitalWrite(LED_PIN3, LOW);
digitalWrite(BUZZER PIN, LOW);
```

3. Inside Loop( ) and sensor specific functions and calling of pins and LEDs

```
if (fall == true) {
  digitalWrite(LED_PIN1, HIGH);
  digitalWrite(BUZZER_PIN, HIGH);
```

```
if ( avg_SpO2 < 90 ) {
  digitalWrite(LED_PIN3, HIGH);
  digitalWrite(BUZZER_PIN, HIGH);
  sendEmail_SpO2(avg_SpO2);
  resetFunc();
}
if (avg_Hr > 170 || avg_Hr < 50) {
  digitalWrite(LED_PIN3, HIGH);
  digitalWrite(BUZZER_PIN, HIGH);
  sendEmail(avg_Hr, avg_SpO2, false);
  resetFunc();
}
if (avg_SpO2 > 100 || avg_Hr < 80) {
  digitalWrite(LED_PIN3, HIGH);
  digitalWrite(BUZZER_PIN, HIGH);
```

```
  if (Avg_Gas > threshold) {
    digitalWrite(LED_PIN2, HIGH);
    digitalWrite(BUZZER_PIN, HIGH);
    sendEmail_Gas(Avg_Gas, true);
```

As you can see PIN1 is used for fall detection, PIN2 is used for Gas detection and PIN3 is used for health alerts only.

Chapter 8: Final Circuit & PCB Fabrication

8.1 BREADBOARD IMPLEMENTATION OF THE CIRCUIT AND I2C PROTOCOL

All the components explained above have been used in the breadboard implementation of the circuit and it has been achieved with the proper understanding of I2C protocol and how delay affects and works in hardware projects. The management of delay in this project is a crucial task as mismanagement of delay can lead to loss of acknowledgment bit which comes from sensors like MAX30100 and MPU6050. If the acknowledgment bit is lost, one of the components will goes out of clock and it will start giving zero values. The same error happens after sending a mail so following reset function is used to restart the microcontroller again which clear all the buffers.

```
void (*resetFunc)(void) = 0;
```

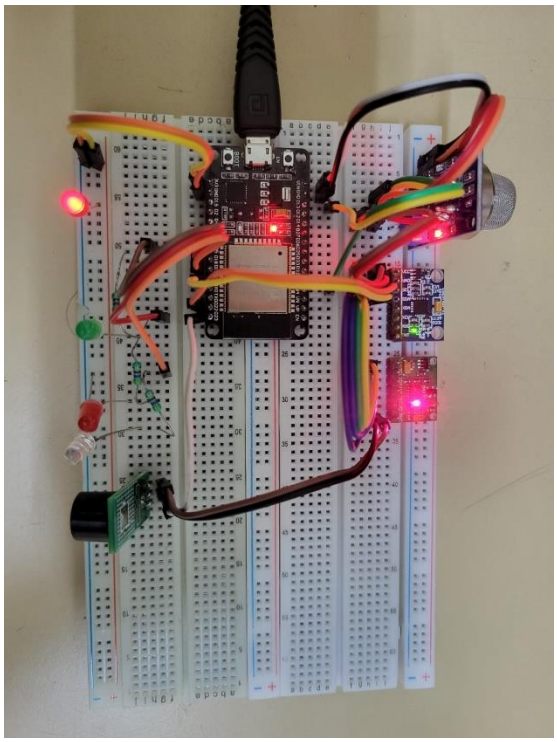

Fig. Breadboard implementation of the circuit

8.2 PCB FABRICATION

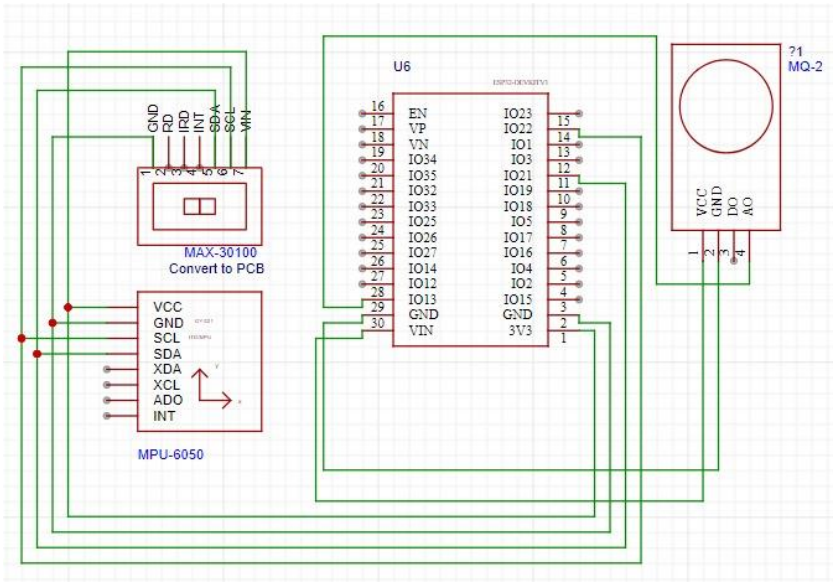

Fig. Schematic Diagram of the PCB layout

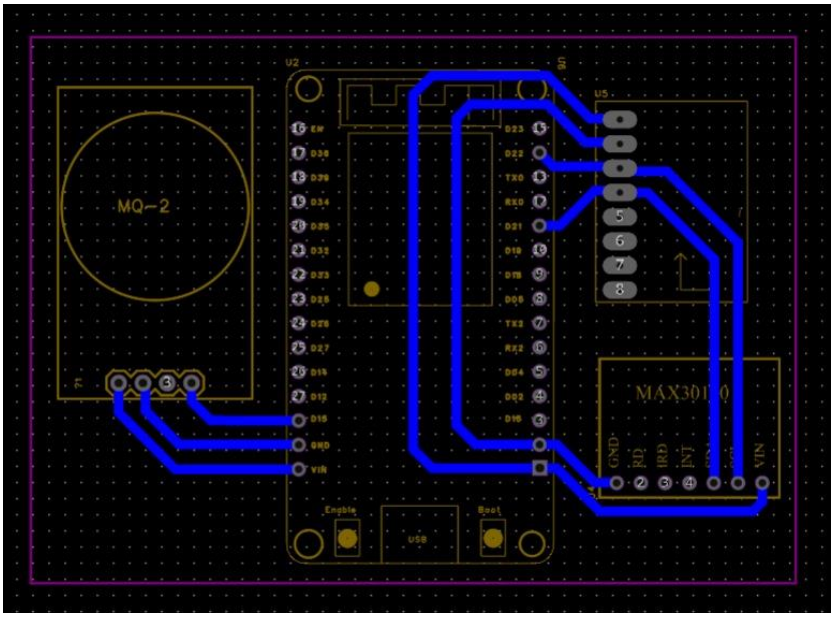

Fig. PCB Layout (PCB Length = 80 mm & Width = 57 mm)

PCB layout has been realized using easyEDA software with a connection thickness of 1 mm on the bottom layer of the PCB while all the components are present on the top layer of the PCB. The components in the PCB are ESP32, MQ-2, MAX30100, and MPU6050.

Chapter 9: Result and Conclusion

9.1 RESULT

The integration of an IoT-based safety monitoring system using ESP-32,MQ-2 gas sensor, MPU6050 accelerometer, and MAX30100 pulse oximeter brings about notable advancements in ensuring the security and health of individuals, especially in environments where gas detection, fall detection, and vital sign monitoring are crucial. This solution amalgamates the capabilities of the ESP32 microcontroller, MQ-2 gas sensor, MPU6050 accelerometer, and MAX30100 pulse oximeter, enhancing the overall safety infrastructure.

9.2 KEY RESULTS AND BENEFITS

Real-time Fall Detection:

The implemented fall detection algorithm, utilizing data from the accelerometer, allows for real-time monitoring of user movements. The ESP-32 processes this data locally, enabling swift detection of potential falls based on predefined thresholds and patterns.

Adaptive Thresholds and Triggers:

The algorithm employs a multi-stage approach with adaptive thresholds and triggers. It considers amplitude vector changes, acceleration thresholds, and orientation variations to distinguish between normal activities and fall events. This adaptability enhances the accuracy of fall detection while minimizing false positives.

Wireless Connectivity and Remote Configuration:

The use of ESP-32 provides wireless connectivity, eliminating the need for physical connections. This wireless capability allows for remote configuration, enabling updates to the fall detection algorithm or system parameters without the need for on-site interventions. This feature enhances the system's flexibility and adaptability to evolving user needs.

User-Friendly Mobile App Interface:

While not explicitly detailed in the provided code, the implementation suggests the potential for a user-friendly mobile app interface. Such an interface could provide users and caregivers with a centralized platform for monitoring fall events, viewing historical data, and configuring system settings. This enhances user engagement and facilitates seamless interaction with the fall detection system.

Efficient Power Management:

The ESP-32's capabilities extend to power management, ensuring that the fall detection system remains energy-efficient. This is crucial for prolonged use, especially in scenarios where continuous monitoring is essential. Efficient power management contributes to the sustainability and reliability of the system.

In conclusion, the IoT-based fall detection system using ESP-32 demonstrates a holistic approach to addressing the challenges associated with fall detection. By leveraging sensor data, adaptive algorithms, and a user-friendly mail system, this solution offers a robust and responsive framework for improving the safety and care of individuals vulnerable to falls. As technology continues to advance, such systems hold promise for broader adoption and integration into healthcare and assisted living environments.

9.3 CONCLUSION

In the grand tapestry of technological innovation, the synergy of ESP-32, MAX30100 pulse oximeter, MPU6050 accelerometer, and MQ-2 gas sensor in the IoT-based fall detection system not only champions real-time safety but also introduces several key features that redefine user-centric care.

Advanced Sensor Fusion: The system's capacity to seamlessly integrate data using I2C protocol from diverse sensors demonstrates advanced sensor fusion. By harnessing the collective power of the pulse oximeter, accelerometer, and gas sensor, the fall detection algorithm gains a

multidimensional understanding of the user's health status, providing a more comprehensive safety net.

Multi-modal Alerts: In addition to conventional alerts, the system's potential for multi-modal notifications, such as vibrations or visual cues, enhances accessibility. This feature caters to users with varying needs, ensuring that alerts are not only timely but also tailored to individual preferences and sensory capabilities.

Environmental Monitoring with Gas Sensor: The inclusion of the MQ-2 gas sensor introduces environmental monitoring capabilities. Beyond fall detection, the system can identify and alert users to the presence of potentially harmful gases, adding an extra layer of safety and well-being to the overall functionality.

Scalability and Interoperability: The modular architecture of the system, accommodating multiple sensors, positions it as a scalable solution. It not only adapts to the current user's needs but also opens avenues for future expansions and interoperability with emerging sensor technologies, keeping the system relevant amid evolving healthcare landscapes.

In summation, the IoT-based fall detection system, with its advanced sensor fusion, machine learning prospects, multi-modal alerts, environmental monitoring, and scalability, emerges as a pioneering force in redefining user safety and well-being. As it continues to evolve, the system not only meets the present demands but also anticipates and prepares for the future horizons of healthcare technology.

9.4 FUTURE WORK

Let's expand on the future work and potential enhancements related to the integration of a smart hard helmet and various sensors for construction worker safety:

1. Real-Time Biometric Monitoring:

- Integrate additional biometric sensors, such as skin temperature and perspiration sensors, to monitor the physiological well-being of workers in real time. This data could offer insights into potential health issues or signs of heat stress, ensuring a proactive approach to worker safety.

2. Predictive Analytics for Health and Safety:

- Implement machine learning algorithms to analyse historical data and predict potential safety hazards or health risks on construction sites. This could enable preventive measures and proactive interventions to mitigate risks before they escalate.

3. Fall Prediction Algorithms:

- Develop advanced fall prediction algorithms by combining data from accelerometers, gyroscopes, and environmental sensors. By identifying patterns leading to falls, the system could provide warnings or suggestions to workers, enhancing their awareness and reducing the likelihood of accidents.

4. AR-Assisted Training and Guidance:

- Leverage augmented reality (AR) for on-site training modules and real-time guidance. Workers could receive step-by-step instructions or visual cues through their helmets, improving task efficiency and ensuring that safety protocols are adhered to.

5. Multi-Sensor Fusion for Enhanced Accuracy:

- Explore the fusion of data from multiple sensors, including cameras, LiDAR, and ultrasonic sensors, to enhance the accuracy of detecting obstacles, hazardous materials, or potential collisions. This multi-sensor approach could create a more comprehensive safety net.

6. Collaborative Safety Ecosystem:

- Foster collaboration among different construction sites by creating a connected safety ecosystem. Shared data on safety incidents, best practices, and lessons learned could contribute to industry-wide improvements in safety standards.

7. Energy-Efficient Hardware Design:

- Investigate energy-efficient hardware designs and power management strategies to prolong the operational life of the smart helmet. This could involve the integration of energy harvesting technologies or advanced battery systems.

8. Adaptive User Interfaces:

- Develop adaptive user interfaces on the helmet's display, considering factors such as ambient lighting conditions and the user's cognitive load. This ensures that information is presented in the most effective and least distracting manner.

9. Regulatory Compliance Monitoring:

- Create features within the system to monitor and ensure compliance with safety regulations and standards. Automated reporting and documentation could assist construction companies in demonstrating adherence to safety guidelines.

10. Human-Machine Collaboration:

- Explore ways in which the smart helmet can facilitate improved communication and collaboration between human workers and autonomous machinery on construction sites. This could enhance overall efficiency and safety in construction processes.

11. User Feedback Integration:

- Establish mechanisms for continuous user feedback and improvement. Workers' experiences and suggestions can provide invaluable insights into the practicality and usability of the system, driving iterative enhancements.

12. Integrated Smart Visor for Enhanced Visibility and Safety:

- Explore the integration of a smart visor into the helmet design, equipped with features such as augmented reality (AR) overlays for displaying critical information, including real-time sensor data, safety alerts, and navigational guidance. The smart visor could also incorporate transparent displays to enhance visibility in challenging environments, providing workers with a clear line of sight while maintaining access to pertinent data. This integration aims to improve situational awareness, allowing workers to focus on their tasks while staying informed about potential hazards or changes in the work environment. Additionally, the smart visor could include anti-glare and tint adjustment functionalities for optimized visibility in varying light conditions, further contributing to overall safety and efficiency on construction sites.

By embracing these future directions, the smart hard helmet evolves beyond a safety tool to become an intelligent and adaptive companion for construction workers, promoting a culture of proactive safety and well-being in the construction industry.

Chapter 10: REFERENCES

[1] S. Peek, "Why your Construction Company Needs Smart Helmet," Business, Mar. 23, 2023. [Online]. Available: https://www.business.com/articles/smart-helmet-construction/. [Accessed: Oct. 5, 2023].

[2] V. Jayasree and M. N. Kumari, "IOT Based Smart Helmet forConstruction Workers," 2020 7th International Conference on Smart Structures and Systems (ICSSS), Chennai, India, 2020, pp. 1-5, doi: 10.1109/ICSSS49621.2020.9202138.

[3] Kai-Stefan Schober(2019, January 9).*Introducing helmets with IoT technologies for construction sites*, Roland Berger. Accessed on:Oct.5,2023.[Online].Available: https://www.rolandberger.com/en/Insights/Publications/Introducing- helmets-with-IoT-technologies-for-construction-sites.html

[4] Kuhar, Preeti, Kaushal Sharma, Yaman Hooda, and Neeraj Kumar Verma."Internet of Things (IoT) Based Smart Helmet for Construction." *Journal of Physics: Conference Series* 1950.1 (2021): 012075.