NITTE (Deemed to be University) | NMAM INSTITUTE OF TECHNOLOGY

Internet of Things Laboratory (EC2601-1)

A Project report on

Distress Detection System

Submitted By

S Vaishali Pai	NNM22EC138
Sahana Prabhu	NNM22EC139
Sai Janani	NNM22EC140
Salman Farish	NNM22EC141

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

May 2025

NITTE (Deemed to be University) | NMAM INSTITUTE OF TECHNOLOGY

Department of Electronics and Communication Engineering

CERTIFICATE

This is to certify that **S Vaishali Pai (NNM22EC138), Sahana Prabhu (NNM22EC139), Sai Janani (NNM22EC140), Salman Farish (NNM22EC141),** bonafide students of N.M.A.M. Institute of Technology, Nitte have submitted the report for the project entitled "**Distress Detection System**" in partial fulfillment of the requirements for the Internet of Things Laboratory (EC2601-1) during the year 2024-2025

Project Evaluation

Name of the Examiners	Signature with date
1. ______________________________	______________________________
2. ______________________________	______________________________

ABSTRACT

This project presents an IoT-based Distress Detection System utilizing the ESP32 microcontroller integrated with physiological sensors and wireless communication to detect and respond to emergency health situations. The system continuously monitors the user's heart rate using the MAX30102 pulse oximeter sensor and stress levels via a GSR (Galvanic Skin Response) sensor. A panic button is also included for manual distress signaling.

The system evaluates real-time sensor data to detect abnormal conditions—such as elevated heart rate or stress levels or user-initiated panic via the button. Upon detecting such a condition, a 10-second pre-alert phase is initiated, allowing the user to cancel the alert via a mobile interface using Blynk IoT. If not cancelled, a final alert is sent via SMS using the Twilio API to a predefined emergency contact (e.g., a doctor). The system also features a 30-second cooldown period after each alert to prevent redundant messages.

This smart and responsive system enables rapid remote health monitoring and emergency alerting, making it highly applicable for elderly individuals, patients with chronic conditions, or individuals working in high-stress or isolated environments.

.

TABLE OF CONTENTS

Chapter	Title		Page No.
	Abstract		1
	Table of Contents		2
	List of Figures		3
1	Introduction		4 - 5
	1.1	Background And motivation	4
	1.2	Problem statement	4
	1.3	Scope of the project	4
	1.4	Objectives	5
	1.5	Significance of the project	5
	1.6	Limitations	5
2	Methodology		6 – 9
	2.1	Flow chart	6
	2.2	Block diagram	7
	2.3	Working Principle	8 - 9
3	Hardware and Software Requirements		10-13
	3.1	Hardware Requirements	10-11
	3.2	Software Requirements	12-13
4	Results and Discussion		14-21
	Conclusion		22
	References		23

List of Figures

Figure 2.1: Flow chart Distress Detection System ..…………………………………6

Figure 2.2: Block diagram of Distress Detection System.……………………………7

Figure 4.1: App Window………………………………………………………………………20

Figure 4.2: Pre-alert notification ……………………..……………………………………20

Figure 4.3: Message from Twilio to doctor guardians ……..………………………..21

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND AND MOTIVATION

In emergency situations, such as sudden illness, panic attacks, or accidents, individuals may be unable to call for help, which can delay critical medical attention and increase risk. This concern is especially relevant for elderly people, patients with heart conditions, and those who work or live alone. With advancements in sensor technology and wireless communication, it is now possible to continuously monitor vital signs like heart rate and stress levels and detect abnormal conditions without direct user input. The increasing availability of IoT platforms and cloud-based services further enables seamless real-time alerting to caregivers or medical personnel. This project is inspired by the need to create a practical, cost-effective solution that bridges the gap between health monitoring and emergency response, ensuring that help can reach those in distress even when they are unable to act on their own

1.2 PROBLEM STATEMENT

In emergency situations, individuals may be unable to seek help due to physical or emotional limitations. Elderly people and those with medical conditions are especially vulnerable. Traditional monitoring systems often lack real-time, automated alert capabilities. Delays in detecting distress can lead to severe health risks or fatalities. There is a need for a system that can monitor vital signs and emotional stress continuously. It should detect abnormal patterns and trigger alerts without manual input. Remote alerting through mobile or cloud services is essential for timely intervention. This project addresses these challenges with an IoT-based automatic distress detection system .

1.3 SCOPE OF THE PROJECT

The project focuses on designing an IoT-based distress detection system that monitors vital signs like heart rate and stress levels in real-time. It includes a panic button for manual distress signaling and uses the ESP32 microcontroller for sensor interfacing and communication. The system sends automated alerts via SMS to emergency contacts when abnormal conditions are detected. The solution leverages cloud services for remote monitoring and control, aiming to provide timely assistance to vulnerable individuals. The scope also includes real-time data transmission, alert management, and mobile app integration.

1.4 OBJECTIVES

1. To enhance quick medical intervention in case of distress and allowing healthcare professionals to respond promptly and effectively.

2. To enable the users to continuously monitor their physiological parameters in real-time using integrated sensors

3. Analyse and make informed decisions about their health .

1.5 SIGNIFICANCE OF THE PROJECT

The project enhances emergency response by enabling real-time detection of distress signals, ensuring timely intervention. It provides continuous health monitoring, offering peace of mind to caregivers and families. The system improves personal safety by detecting distress even when individuals cannot seek help. It gives users the ability to manage alerts, ensuring both safety and control.

1.6 LIMITATIONS

1. Reliance on Sensor Accuracy : The system's effectiveness depends on the accuracy and reliability of the sensors (heart rate, GSR, etc.), which may be affected by external factors such as user movement, skin conditions, or sensor placement.

2. Connectivity Issues : The system requires a stable Wi-Fi connection for cloud communication and alert notifications. Any disruption in internet connectivity could delay or prevent alerts from being sent.

3. False Alerts : The system may generate false alerts due to sensor errors, environmental factors, or incorrect user inputs. For example, during intense physical activity like exercising in the gym, heart rate can increase naturally, which might trigger an unnecessary alert.

4 . Limited Coverage : The system relies on specific sensors, which may not detect all emergencies, such as physical injuries or environmental hazards. It also may not differentiate between normal physical exertion and a real medical emergency.

CHAPTER 2

METHODOLOGY

The methodology involves using an ESP32 microcontroller to interface with heart rate, GSR, and panic button sensors for continuous monitoring. The data is processed in real time to detect abnormal patterns indicating distress. When distress is detected, the system triggers an alert via SMS through the Twilio API. Additionally, it integrates with the Blynk platform for remote monitoring and control via a mobile app.

2.1 FLOW CHART

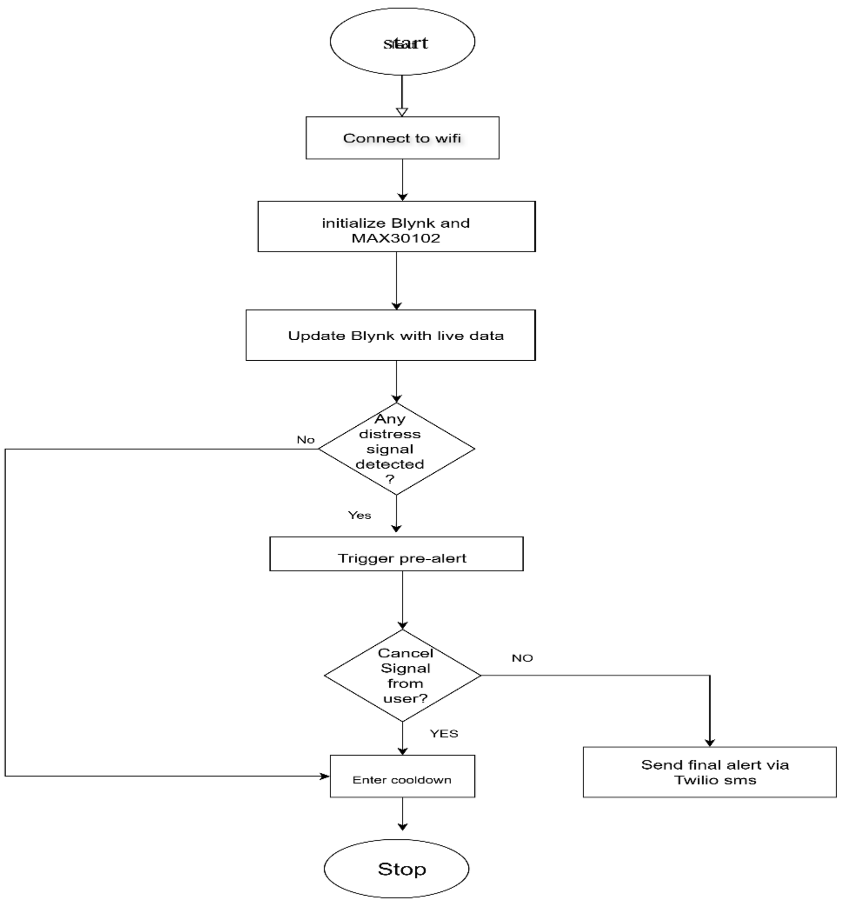

start
Connect to wifi
initialize Blynk and MAX30102
Update Blynk with live data
Any distress signal detected ?
No
Yes
Trigger pre-alert
Cancel Signal from user?
NO
YES
Send final alert via Twilio sms
Enter cooldown
Stop

Fig 2.1 : Flow chart of Distress Detection System

The flowchart represents a health or emergency monitoring system that utilizes a MAX30102 sensor, the Blynk IoT platform, and Twilio SMS service to detect and respond to distress signals. The process begins with the system connecting to Wi-Fi, followed by the initialization of the Blynk platform and the MAX30102 sensor, which monitors vital signs such as heart rate and blood oxygen levels. Live sensor data is continuously updated to the Blynk app for real-time tracking. The system then checks for any signs of distress. If a distress signal is immediately detected, a final alert is sent via Twilio SMS to emergency contacts. If no distress is detected, a pre-alert is triggered, giving the user an opportunity to cancel the signal in case it is a false alarm. If the user cancels the alert, the system enters a cooldown phase to prevent repeated alerts. If not canceled, a final emergency message is sent via SMS. The process concludes with the system stopping or resetting, ensuring efficient and safe monitoring of the user's health status

2.2 BLOCK DIAGRAM

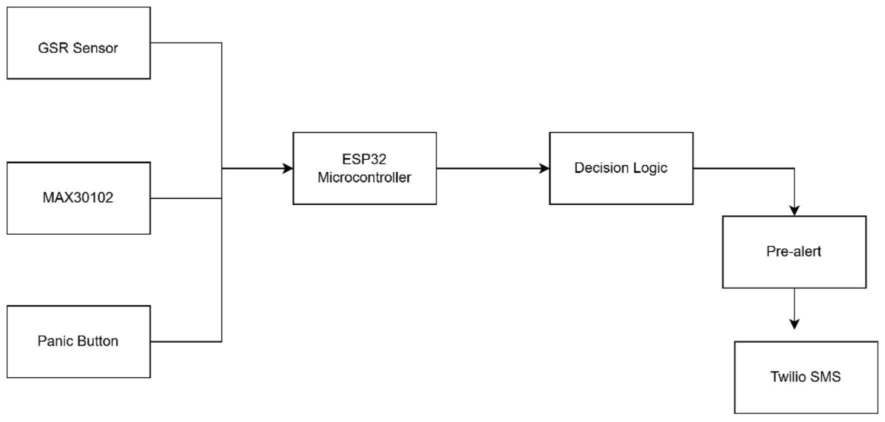


Fig 2.2 : Block diagram of Distress Detection System

The block diagram illustrates a Distress Detection and Alert System centered around an ESP32 microcontroller, which collects input from a GSR sensor (to detect stress via skin resistance), a MAX30102 sensor (to monitor heart rate and oxygen levels), and a panic button (for manual emergency activation). These sensor readings are processed by the ESP32 using a well-defined decision logic, which plays a critical role in determining whether to initiate an alert. This logic first checks if any of the three inputs indicate potential distress: if the GSR value exceeds a threshold (e.g., 2500, suggesting stress), if the average heart rate (BPM) is abnormally high (e.g., over 120 BPM, indicating physical or emotional stress), or if the panic button is pressed. If any of these conditions are met, the system triggers a pre-alert phase and starts a 10-second countdown, giving the user a chance to cancel the alert using the Blynk app (via a virtual button). During this period, the system monitors the cancel State variable; if the user intervenes, the alert is aborted and the system resets. However, if the user does not cancel within the window, the decision logic escalates the alert and uses Twilio's API to send an emergency SMS to a preconfigured contact. To avoid spamming, the system then enters a cooldown phase (e.g., 30 seconds), during which it ignores further alerts. Throughout operation, the ESP32 sends real-time data (like BPM, GSR, and panic button status) to the Blynk app for remote monitoring, making this system a comprehensive solution for automated and user-controlled distress detection and response .

2.3 WORKING PRINCIPLE

1. Sensor Data Collection

The ESP32 collects real-time data from the GSR sensor, MAX30102 sensor (for heart rate), and panic button.

- The GSR sensor detects stress via changes in skin resistance.
- The MAX30102 sensor monitors heart rate (BPM), detecting physical distress.
- The panic button allows the user to manually trigger an alert.

2. Distress Detection and Pre-Alert Phase

The system checks if any sensor input exceeds predefined distress thresholds.

- If the GSR is too high or heart rate exceeds 120 BPM, distress is detected.
- The panic button also triggers distress.
- A 10-second pre-alert countdown begins, during which the user can cancel the alert using the Blynk app.

3. Alert Activation

If the user does not cancel the alert, an emergency SMS is sent via the Twilio API.

- The SMS notifies emergency contacts of the detected distress.
- The system provides BPM and GSR values for context in the message.
- This ensures immediate action can be taken by the contact.

4. Cooldown Phase and Resumption

After sending the alert, the system enters a 30-second cooldown phase.

- During this time, the system ignores further alerts to prevent false alarms.
- After the cooldown, it resumes normal monitoring, continuously checking for distress.
- The system ensures no repeated alerts during this phase .

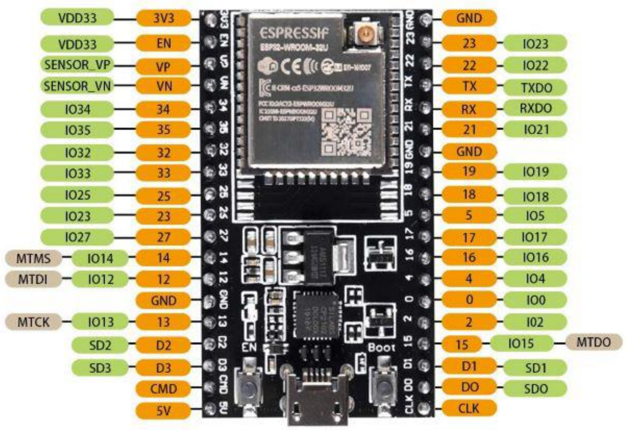
CHAPTER 3

HARDWARE AND SOFTWARE REQUIREMENTS

3.1 Hardware Components:

1. ESP32 Microcontroller

- **Key Points & Specialty**:
  - Dual-core processor for efficient task management.
  - Wi-Fi and Bluetooth capabilities for wireless communication.
  - Low-power consumption, ideal for battery-operated projects.
  - Real-time processing of sensor data and decision-making.
- **Working**:
  The ESP32 continuously collects data from all connected sensors (GSR, heart rate, panic button) and processes them in real-time. It runs the decision logic to evaluate distress and triggers alerts when necessary. Additionally, it handles communication with the Blynk app for remote monitoring and control, and sends emergency SMS using Twilio API when distress is detected.



2. GSR Sensor

- **Key Points & Specialty**:
  - Measures skin electrical resistance to detect stress via sweat.
  - Non-invasive and simple to integrate into wearable devices.

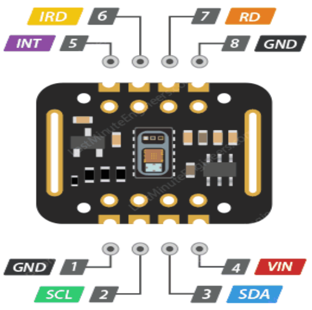
- Effective for emotion or physical stress detection.

- **Working**:
  The GSR sensor measures the resistance of the skin, which changes as the user experiences stress or anxiety (increased sweat). When the resistance falls below a certain threshold, it indicates potential distress. This data is sent to the ESP32 for evaluation, triggering the alert logic if stress levels are detected.

3. MAX30102 Sensor

- **Key Points & Specialty**:
  - Combines IR and Red LEDs with a photodiode for non-invasive measurements.
  - Measures heart rate (BPM) .
  - Compact and low-power, suitable for wearables and health-monitoring devices.

- **Working**:
  The MAX30102 uses light absorption techniques to detect changes in blood flow under the skin. It measures the time between the light emitted and the light reflected to calculate the BPM (heart rate) . These values help assess the user’s physical condition, and if the heart rate exceeds the distress threshold (e.g., 120 BPM), it triggers the alert mechanism.



4.Panic Button

- **Key Points & Specialty**:
    - Simple digital input (HIGH/LOW) for easy activation.
    - Immediate response to trigger an emergency alert manually.
    - User-controlled activation for emergency situations.
- **Working**:
  The panic button is a manual override to trigger the emergency alert. When the button is pressed, it sends a HIGH signal to the ESP32, indicating immediate distress. This bypasses the sensor data and directly triggers the alert mechanism, such as sending an emergency SMS via Twilio API.

3.2 Software Components:

1. Arduino IDE

- **Key Points**:
    - Open-source development environment.
    - Supports coding in C/C++ for ESP32.
    - Easy library integration (GSR, MAX30102, Blynk, Twilio).
    - Serial Monitor for debugging and testing.
- **Working**:
  Arduino IDE is used to write, compile, and upload code to the ESP32, allowing sensor reading, decision logic, and communication to function

2. Blynk Platform

- **Key Points**:
  - Cloud-based IoT interface.
  - Provides real-time monitoring UI.
  - Virtual pins for two-way communication.
  - Mobile app for Android/iOS.
- **Working**:
  Blynk enables the user to monitor sensor data in real-time and control the system remotely, such as canceling an alert through a virtual button.

3. Twilio API

- **Key Points**:
  - RESTful API for sending SMS.
  - Easy to integrate using HTTPS requests.
  - Secure and scalable messaging.
  - Requires API credentials and phone number setup.
- **Working**:
  Twilio API is triggered from the ESP32 to send an emergency SMS when distress is detected and not canceled within the pre-alert period .

CHAPTER 4

RESULTS AND DISCUSSIONS

The implemented distress detection and alert system is designed to continuously and effectively monitor the user's physical and emotional state. It gathers data from multiple sources: the GSR sensor, which detects changes in skin resistance indicating stress levels; the MAX30102 sensor, which measures heart rate and oxygen saturation to monitor physiological changes; and a manual panic button, which provides the user with a direct way to report distress. The core logic of the system evaluates these inputs against predefined thresholds—such as a high heart rate or elevated stress level—to detect signs of potential danger or discomfort.

When any of these conditions are met, the system transitions into a 10-second pre-alert phase, giving the user a chance to acknowledge or override the alert condition. If the user does not cancel the alert within this window, the system treats the situation as an emergency. It then automatically sends an SMS alert using the Twilio API to a predefined emergency contact, including key information such as the user's heart rate and stress levels. This ensures that help can be dispatched quickly and appropriately.

Throughout its operation, the system maintains a constant stream of real-time data collection and communication, ensuring up-to-date monitoring and responsive decision-making. Its design emphasizes automation, accuracy, and user safety, making it a reliable and proactive solution for emergency detection and personal safety management.

ESP-32 code on Arduino IDE

```
#define BLYNK_TEMPLATE_ID "TMPL3tN_SPne5"

#define BLYNK_TEMPLATE_NAME "Distress Detection"

#define BLYNK_AUTH_TOKEN "pJoDLn8QFvPWzf-Kf1dzh50bnOTrlqF8"

#include <WiFi.h>

#include <Wire.h>

#include "MAX30105.h"

#include "heartRate.h"
```

```
#include <BlynkSimpleEsp32.h>

#include <HTTPClient.h>

#include <Base64.h>

#define WIFI_SSID "Airtel_Fariha"

#define WIFI_PASSWORD "8296867161"

#define TWILIO_SID "ACbfe42e4aded3d76706edde29b880e8d6"

#define TWILIO_AUTH_TOKEN "185d4d6c6478992c93635f7a487922df"

#define TWILIO_PHONE_NUMBER "+16282503635"

#define DOCTOR_PHONE_NUMBER "+917349355637"

#define GSR_PIN 34

#define BUTTON_PIN 4

#define GSR_THRESHOLD 2500

#define BPM_THRESHOLD 120

MAX30105 particleSensor;

const byte RATE_SIZE = 10;

byte rates[RATE_SIZE];

byte rateSpot = 0;

long lastBeat = 0;

float beatsPerMinute = 0;

int beatAvg = 0;

bool alertTriggered = false;

bool cancelState = false;

bool inCooldown = false;

unsigned long alertStartTime = 0;

unsigned long cooldownStartTime = 0;

unsigned long lastSendTime = 0;

int gsrValue = 0;

bool buttonPressed = false;
bool fingerDetected = false;

BLYNK_WRITE(V4) {

  cancelState = param.asInt();

  if (cancelState) {

    Serial.println(" Alert Cancelled by User!");

  }

}

void sendTwilioMessage(String message) {

  HTTPClient http;

  String url = "https://api.twilio.com/2010-04-01/Accounts/" + String(TWILIO_SID) + "/Messages.json";

  String auth = String(TWILIO_SID) + ":" + String(TWILIO_AUTH_TOKEN);

  String encodedAuth = base64::encode(auth);

  http.begin(url);

  http.addHeader("Authorization", "Basic " + encodedAuth);

  http.addHeader("Content-Type", "application/x-www-form-urlencoded");

  String body = "To=" + String(DOCTOR_PHONE_NUMBER) + "&From=" + String(TWILIO_PHONE_NUMBER) +
"&Body=" + message;

  int httpResponseCode = http.POST(body);

  if (httpResponseCode > 0) {

    Serial.println("Twilio Message Sent Successfully.");

  } else {

    Serial.print(" Twilio Error. HTTP Code: ");

    Serial.println(httpResponseCode);

  }

http.end();

}

void setup() {

  Serial.begin(115200);

  pinMode(BUTTON_PIN, INPUT_PULLUP);
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("\n WiFi  Has been Connected You can Proceed ");
  Blynk.begin(BLYNK_AUTH_TOKEN, WIFI_SSID, WIFI_PASSWORD);
  if (!particleSensor.begin(Wire, I2C_SPEED_FAST)) {
    Serial.println(" MAX30102  has not not found");
    while (1);
  }
  particleSensor.setup();
  particleSensor.setPulseAmplitudeRed(0x0A);
  particleSensor.setPulseAmplitudeGreen(0);
}
void loop() {
  Blynk.run();
  if (inCooldown && millis() - cooldownStartTime < 30000) return;
  if (inCooldown) {
    inCooldown = false;
    Serial.println(" Cooldown Finished.");
  }
  buttonPressed = digitalRead(BUTTON_PIN) == LOW;
  gsrValue = analogRead(GSR_PIN);
  long irValue = particleSensor.getIR();
  fingerDetected = irValue > 50000;
  if (fingerDetected && checkForBeat(irValue)) {
    long delta = millis() - lastBeat;
    lastBeat = millis();

    beatsPerMinute = 60 / (delta / 1000.0);

    if (beatsPerMinute > 20 && beatsPerMinute < 200) {

      rates[rateSpot++] = (byte)beatsPerMinute;

      rateSpot %= RATE_SIZE;

      beatAvg = 0;

      for (byte i = 0; i < RATE_SIZE; i++) beatAvg += rates[i];

      beatAvg /= RATE_SIZE;

    }

  }

  if (millis() - lastSendTime > 5000) {

    lastSendTime = millis();

    Blynk.virtualWrite(V0, fingerDetected ? beatAvg : 0);

    Blynk.virtualWrite(V1, fingerDetected ? beatsPerMinute : 0);

    Blynk.virtualWrite(V2, gsrValue);

    Blynk.virtualWrite(V3, buttonPressed ? 1 : 0);

    Serial.println("----------- STATUS -----------");

    Serial.print("GSR Value     : "); Serial.println(gsrValue);

    Serial.print("BPM           : "); Serial.println(beatsPerMinute);

    Serial.print("Avg BPM       : "); Serial.println(beatAvg);

   }

  bool gsrAlert = gsrValue > GSR_THRESHOLD;

  bool bpmAlert = beatAvg > BPM_THRESHOLD;

  if ((buttonPressed || gsrAlert || bpmAlert) && !alertTriggered) {

    Serial.println("🚨 Pre-Alert Triggered.");

    Blynk.logEvent("pre_alert");

    alertStartTime = millis();

    alertTriggered = true;
```

```
    cancelState = false;
  }
  if (alertTriggered && millis() - alertStartTime <= 10000) {
    if (cancelState) {
      alertTriggered = false;
      Serial.println("Alert Cancelled.");
    }
}
  if (alertTriggered && millis() - alertStartTime > 10000) {
    if (!cancelState) {
      Serial.println("🚨 Sending Final Alert (SMS)");
      sendTwilioMessage("🚨 Distress detected! Immediate help needed!");
    }
    alertTriggered = false;
    inCooldown = true;
    cooldownStartTime = millis();
  }
}
```

Figure 4.1 : App Window

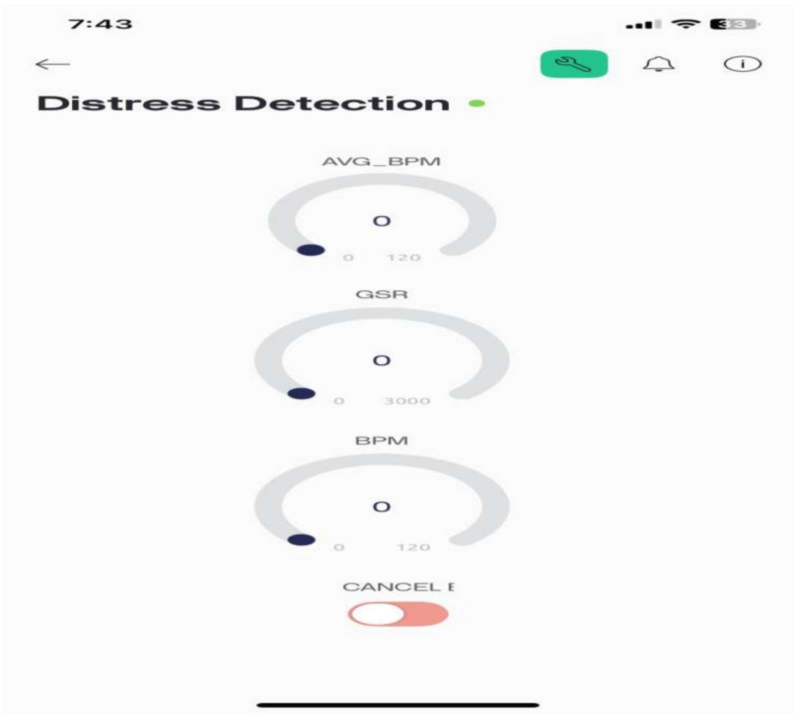
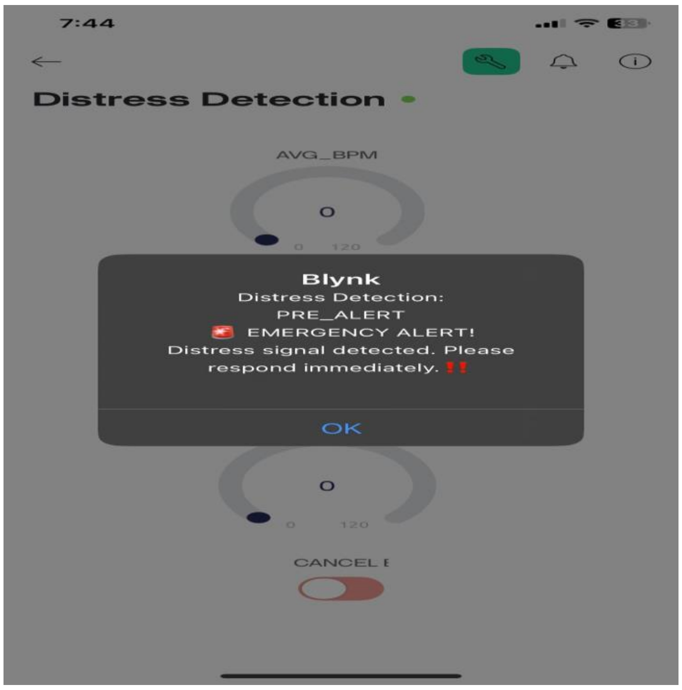
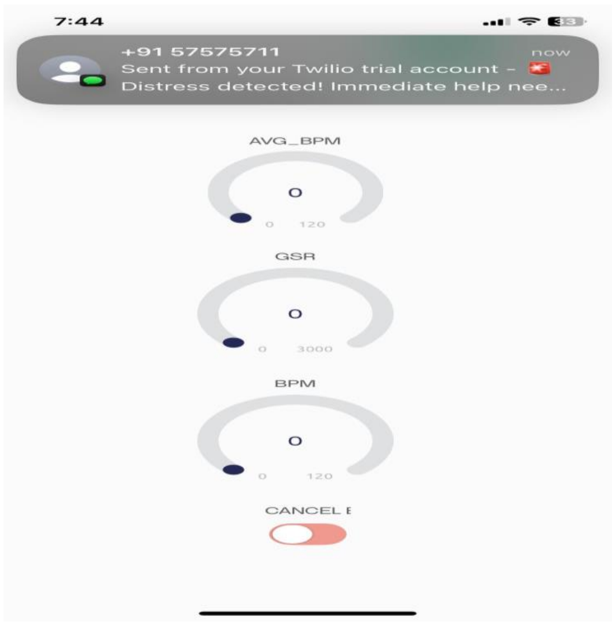


Figure 4.2 : Pre-alert notification


Blynk
Distress Detection:
PRE_ALERT
🚨 EMERGENCY ALERT!
Distress signal detected. Please respond immediately. ‼

OK

Figure 4.1 : Message from Twilio to doctor or guardians


7:44

+91 57575711 now

Sent from your Twilio trial account – 🚨

Distress detected! Immediate help nee...

CONCLUSION

The implemented distress detection and alert system effectively combines sensor technology and IoT capabilities to monitor a user's emotional and physical condition in real-time. It utilizes the GSR sensor to detect stress through skin resistance, the MAX30102 sensor to track heart rate and oxygen levels, and a panic button for manual emergency activation. These inputs are processed by the ESP32 microcontroller using defined threshold values. When distress is detected, the system enters a 10-second pre-alert phase, allowing the user to cancel if it was a false trigger. If no action is taken, an emergency SMS is automatically sent using the Twilio API, ensuring a quick response from trusted contacts.

This project highlights the powerful integration of embedded systems with real-world applications in health and safety. It provides a reliable, user-friendly, and automated method for emergency detection and communication. The inclusion of real-time data monitoring, decision logic, and wireless communication makes the system highly efficient and practical. Future improvements, such as GPS tracking or wearable integration, could further enhance its usability and expand its application in personal safety, elderly care, and high-stress environments.

REFERENCES

[1] **MAX30102 Pulse Oximeter and Heart-Rate Sensor** – Datasheet, Maxim Integrated. Available at: https://datasheets.maximintegrated.com/en/ds/MAX30102.pdf

[2] **GSR Sensor Module – Skin Conductance Sensor** Documentation, Seeed Studio. Available at: https://wiki.seeedstudio.com/Grove-GSR_Sensor/