# CONTENT:

|  |  |  |
| --- | --- | --- |
| **Serial No.** | **Topic** | **Page No.** |
| 1  2  3  4  5  6  7  8  9  10 | Introduction Objective  Functionalities  Sensor Used  Block Diagram  Future Upgradation  Programming Observation Conclusion  Signature | 3  3  4  4  5  5-6  6-13  13-14  14  15 |

## Aim:

To protect the rider from accidents, inform the family and friends of an incident of a crash.

## Introduction:

In an era where technology is seamlessly integrating into every aspect of daily life, smart helmets have emerged as a revolutionary innovation in safety and convenience. Unlike traditional helmets, which focus solely on physical protection, smart helmets incorporate advanced features such as built-in communication systems, GPS navigation, augmented reality (AR) displays, and impact sensors. These helmets are designed to enhance safety, improve user experience, and provide real-time assistance, making them ideal for motorcyclists, cyclists, construction workers, and even military personnel. With the increasing need for smarter safety solutions, smart helmets represent the future of protective gear, offering a blend of innovation, comfort, and cutting-edge technology.

## Objective:

The primary objective of a smart helmet is to enhance safety, communication, and convenience for users by integrating advanced technology into traditional protective headgear. Specifically, the smart helmet aims to:

1. **Improve Safety** – Detect accidents through impact sensors and send emergency alerts to predefined contacts or emergency services.
2. **Enhance Communication** – Enable hands-free calling and voice-assisted controls for seamless connectivity while on the move.
3. **Provide Navigation Assistance** – Incorporate GPS navigation with voice guidance or heads-up display (HUD) for safer and distraction-free travel.
4. **Monitor Health & Environment** – Include sensors to track vital signs, fatigue levels, and environmental conditions like air quality or temperature.
5. **Increase Visibility & Awareness** – Integrate smart lighting, rear-view cameras, or AR displays to improve situational awareness and prevent accidents.
6. **Promote Compliance & User Comfort** – Ensure helmets are worn properly using smart locking mechanisms or compliance monitoring features.

By achieving these objectives, smart helmets aim to redefine personal safety and convenience across various industries, including transportation, construction, and defense.

## Functionalities:

### ALCOHOL DETECTION:

Detects alcohol, sends a warning message via gmail, autocuts the start of the bike, displays message in OLED display.

### CRASH DETECTION:

If a strong vibration is received and a change in axis is detected using gyroscope, a distress alarm rings in family or friends phone and an email is sent. A link is sent which can be used to look after the situation through a camera setup on the helmet

### WEAR DETECTION:

.

The bike does not start until the helmet is worn.

### ACCIDENT PROBABILITY CALCULATION:

If the proximity of the vehicles or other objects are very near, accident prone and look left, right is displayed in OLED.

**Sensor Used:**

1. MQ3(Alcohol detection)
2. I2C OLED display - 2
3. MP06050 accelerator + gyroscope
4. FSR sensor
5. Relay
6. Ultrasonic sensors - 3
7. ESP 32 module - 2
8. ESP 32 cam

### PROGRAMMING LANGUAGE USED:

## Micro python

## Block Diagrams:

SMART

HELMET

WIFI

ENABLED

BIKE

ESP32

WARNING

MESSAGE

VIA API

### FUTURE UPGRADABILITY:

A machine learning model can be applied using computer vision through the ESP 32 cam to identify road signs and potential risks, further reducing risk of accidents.

input

output

ESP 32

(helmet)

IC2 OLED

EMAIL

AND

WARNING

ESP 32 IN

BIKE

OLED

DISPLAY

RELAY

ESP32

CAM

FSR

MA3

3

ULTRA-

SONIC

MP06050

## Programming:

**CODE 1**

from machine import Pin, I2C, ADC, time\_pulse\_us, SoftI2C

import ssd1306

from time import sleep

import time

import math

import network

import mrequests

import socket

# Global variables

digital\_value = 0

total\_accel = 0

gyro\_x, gyro\_y, gyro\_z = 0, 0, 0

fsr\_value = 0

vibration\_value = 0

distance = 0

ACCEL\_THRESHOLD = 3.0

GYRO\_THRESHOLD = 180.0

# I2C for MPU6050

i2c\_mpu = I2C(0, scl=Pin(22), sda=Pin(21)) # Hardware I2C

# I2C for OLED

i2c\_oled = SoftI2C(scl=Pin(25), sda=Pin(26))

oled\_width, oled\_height = 128, 64

oled = ssd1306.SSD1306\_I2C(oled\_width, oled\_height, i2c\_oled)

# Wi-Fi credentials

SSID = "Archismans23fe"

PASSWORD = "archisman2004"

# Server IP and port

SERVER\_IP = "192.168.76.162" # Replace with the server's IP address

SERVER\_PORT = 80

def connect\_wifi():

wlan = network.WLAN(network.STA\_IF)

wlan.active(True)

wlan.connect(SSID, PASSWORD)

print("Connecting to Wi-Fi...")

while not wlan.isconnected():

pass

print("Connected to Wi-Fi!")

print(wlan.ifconfig())

# Blynk credentials

BLYNK\_TOKEN = "AnOPZqT7enN6fdXKwyJ5aCfZF6\_w6hzc"

EVENT\_CODE1 = "smart\_helmet"

url1 = f"http://blynk.cloud/external/api/logEvent?token={BLYNK\_TOKEN}&code={EVENT\_CODE1}"

EVENT\_CODE2="alcohol\_detected"

url2 = f"http://blynk.cloud/external/api/logEvent?token={BLYNK\_TOKEN}&code={EVENT\_CODE2}"

def trigger\_event():

try:

response = mrequests.get(url1)

print("Response Code:", response.status\_code)

print("Response Text:", response.text)

response.close()

except Exception as e:

print("Error while sending request:", e)

def trigger\_warning():

try:

response = mrequests.get(url2)

print("Response Code:", response.status\_code)

print("Response Text:", response.text)

response.close()

except Exception as e:

print("Error while sending request:", e)

def trigger\_event\_with\_retry(retries=3):

for attempt in ranger(,,retries):

try:

trigger\_event()

break # Exit loop if successful

except Exception as e:

print(f"Attempt {attempt + 1} failed: {e}")

time.sleep(2)

else:

print("All attempts to trigger event failed.")

def send\_trigger(trigger\_type):

"""

Sends a trigger message to the server.

trigger\_type: str ("TRIGGER1" to turn ON, "TRIGGER2" to turn OFF)

"""

addr = socket.getaddrinfo(SERVER\_IP, SERVER\_PORT)[0][-1]

client = socket.socket()

client.connect(addr)

print("Connected to server:", addr)

request = "GET /{} HTTP/1.1\r\nHost: {}\r\n\r\n".format(trigger\_type, SERVER\_IP)

client.send(request.encode("utf-8"))

print(f"{trigger\_type} message sent!")

response = client.recv(1024).decode("utf-8")

print("Response from server:", response)

client.close()

# MQ3 Alcohol Detection

def alcohol():

global digital\_value

mq3\_digital\_pin = Pin(15, Pin.IN) # GPIO15 as input

digital\_value = mq3\_digital\_pin.value()

return digital\_value

# MPU6050 Crash Detection

def gyroscope\_accelerometer():

global total\_accel, gyro\_x, gyro\_y, gyro\_z

MPU6050\_ADDR = 0x68

PWR\_MGMT\_1 = 0x6B

ACCEL\_XOUT\_H = 0x3B

GYRO\_XOUT\_H = 0x43

def init\_mpu6050():

i2c\_mpu.writeto\_mem(MPU6050\_ADDR, PWR\_MGMT\_1, b'\x00') # Wake up MPU6050

def read\_word(register):

high, low = i2c\_mpu.readfrom\_mem(MPU6050\_ADDR, register, 2)

value = (high << 8) | low

return value - 65536 if value > 32767 else value

def read\_acceleration():

ax = read\_word(ACCEL\_XOUT\_H) / 16384.0

ay = read\_word(ACCEL\_XOUT\_H + 2) / 16384.0

az = read\_word(ACCEL\_XOUT\_H + 4) / 16384.0

return ax, ay, az

def read\_gyroscope():

gx = read\_word(GYRO\_XOUT\_H) / 131.0

gy = read\_word(GYRO\_XOUT\_H + 2) / 131.0

gz = read\_word(GYRO\_XOUT\_H + 4) / 131.0

return gx, gy, gz

init\_mpu6050()

accel\_x, accel\_y, accel\_z = read\_acceleration()

gyro\_x, gyro\_y, gyro\_z = read\_gyroscope()

total\_accel = math.sqrt(accel\_x\*2 + accel\_y2 + accel\_z\*2)

# FSR Sensor Detection

def read\_fsr():

global fsr\_value,fsr\_value1

fsr\_pin = ADC(Pin(36))

fsr\_pin1 = ADC(Pin(34))

fsr\_pin.atten(ADC.ATTN\_11DB)

fsr\_pin1.atten(ADC.ATTN\_11DB)

fsr\_value = fsr\_pin.read()

fsr\_value1 = fsr\_pin1.read()

return fsr\_value

return fsr\_value1

# Vibration Sensor Detection

def read\_analog\_vibration():

global vibration\_value

vibration\_pin = ADC(Pin(33))

vibration\_pin.atten(ADC.ATTN\_11DB)

vibration\_value = vibration\_pin.read()

if vibration\_value > 3000:

print("Strong vibration detected!")

elif vibration\_value > 1000:

print("Mild vibration detected!")

def ultrasonic():

TRIG\_PIN = 5

ECHO\_PIN = 18

TRIG\_PIN\_LEFT = 2

ECHO\_PIN\_LEFT = 4

TRIG\_PIN\_RIGHT = 13

ECHO\_PIN\_RIGHT = 12

trigleft = Pin(TRIG\_PIN\_LEFT, Pin.OUT)

echoleft = Pin(ECHO\_PIN\_LEFT, Pin.IN)

trigright = Pin(TRIG\_PIN\_RIGHT, Pin.OUT)

echoright = Pin(ECHO\_PIN\_RIGHT, Pin.IN)

trig = Pin(TRIG\_PIN, Pin.OUT)

echo = Pin(ECHO\_PIN, Pin.IN)

global distance, distanceleft, distanceright

def measure\_distanceleft():

trigleft.value(0)

time.sleep\_us(2)

trigleft.value(1)

time.sleep\_us(10)

trigleft.value(0)

pulse\_time = time\_pulse\_us(echoleft, 1, 50000)

if pulse\_time < 0:

return None

return (pulse\_time / 2) / 29.1

def measure\_distanceright():

trigright.value(0)

time.sleep\_us(2)

trigright.value(1)

time.sleep\_us(10)

trigright.value(0)

pulse\_time = time\_pulse\_us(echoright, 1, 50000)

if pulse\_time < 0:

return None

return (pulse\_time / 2) / 29.1

def measure\_distance():

trig.value(0)

time.sleep\_us(2)

trig.value(1)

time.sleep\_us(10)

trig.value(0)

pulse\_time = time\_pulse\_us(echo, 1, 50000)

if pulse\_time < 0:

return None

return (pulse\_time / 2) / 29.1

# Measure distances

distanceleft = measure\_distanceleft()

time.sleep(0.1)

distanceright = measure\_distanceright()

time.sleep(0.1)

distance = measure\_distance()

# Main Loop

receiver\_name = "ESP32\_Receiver"

connect\_wifi()

while True:

try:

alcohol()

read\_fsr()

gyroscope\_accelerometer()

read\_analog\_vibration()

ultrasonic()

time.sleep(0.3)

oled.fill(0) # Clear the screen

if fsr\_value == 4095:

oled.text('Helmet Worn', 0, 0)

oled.show()

send\_trigger\_with\_retry("TRIGGER1")

print("Helmet worn")

if digital\_value == 0:

oled.text('Bike Won\'t Start!', 0, 20)

oled.text('Alcohol Detected!', 0, 30)

oled.show()

send\_trigger\_with\_retry("TRIGGER2")

trigger\_warning\_with\_retry()

# Crash detection logic

if (total\_accel > ACCEL\_THRESHOLD or

abs(gyro\_x) > GYRO\_THRESHOLD or

abs(gyro\_y) > GYRO\_THRESHOLD or

abs(gyro\_z) > 300) and vibration\_value > 3000:

oled.text('Crash Detected!', 0, 50)

oled.text('CALL 100,102!', 0, 60)

oled.show()

trigger\_event\_with\_retry()

# Rest of your sensor logic...

except Exception as e:

print("Error in main loop:", e)

code for circuit in helmet

**CODE 2**

import network

import socket

import machine

import time

# Configure Wi-Fi

SSID = "Archismans23fe"

PASSWORD = "archisman2004"

# Configure Relay

RELAY\_PIN = 2

relay = machine.Pin(RELAY\_PIN, machine.Pin.OUT)

# Set relay to default state (ON)

relay.value(1) # Assume active-low relay; set HIGH to keep it ON initially

# Flag to track the state of the system

last\_trigger = None # Tracks the last trigger received

permanent\_stop = False # Flag to indicate if the relay should be stopped permanently

def connect\_to\_wifi():

"""

Connects to Wi-Fi.

"""

wlan = network.WLAN(network.STA\_IF)

wlan.active(True)

wlan.connect(SSID, PASSWORD)

print("Connecting to Wi-Fi...")

while not wlan.isconnected():

pass

print("Connected! IP address:", wlan.ifconfig()[0])

def handle\_trigger(request):

"""

Handles triggers based on the received HTTP request.

"""

global last\_trigger, permanent\_stop

if "TRIGGER1" in request:

print("TRIGGER1 received! Turning ON the relay.")

relay.value(0) # Turn ON the relay (LOW for active-low relay)

last\_trigger = "TRIGGER1"

permanent\_stop = False # Reset permanent stop if TRIGGER1 is received

elif "TRIGGER2" in request:

print("TRIGGER2 received! Turning OFF the relay.")

relay.value(1) # Turn OFF the relay (HIGH for active-low relay)

last\_trigger = "TRIGGER2"

# First wait for 10 seconds

print("Waiting 10 seconds for another TRIGGER2...")

time.sleep(10) # Wait for 10 seconds

if last\_trigger != "TRIGGER2":

print("No additional TRIGGER2 received in 10 seconds. Turning ON the relay again.")

relay.value(0) # Turn ON the relay (LOW for active-low relay)

else:

# Enter 30-second waiting period

print("Another TRIGGER2 received! Waiting 30 seconds...")

time.sleep(30) # Wait for 30 seconds

if last\_trigger != "TRIGGER2":

print("No additional TRIGGER2 received in 30 seconds. Turning ON the relay again.")

relay.value(0) # Turn ON the relay (LOW for active-low relay)

else:

# Stop the relay permanently

print("Another TRIGGER2 received within 30 seconds. Stopping relay permanently.")

permanent\_stop = True

relay.value(1) # Keep relay OFF (HIGH for active-low relay)

else:

print("Unknown request. Relay state unchanged.")

def start\_server():

"""

Starts the HTTP server to listen for incoming requests.

Controls the relay based on received trigger messages.

"""

addr = socket.getaddrinfo("0.0.0.0", 80)[0][-1]

server = socket.socket()

server.bind(addr)

server.listen(1)

print("Server listening on", addr)

while True:

conn, client\_addr = server.accept()

print("Connection from", client\_addr)

request = conn.recv(1024).decode("utf-8")

print("Request received:", request)

if not permanent\_stop: # Check if the relay is permanently stopped

handle\_trigger(request)

else:

print("Relay permanently stopped. Ignoring all triggers.")

# Send an HTTP response

response = "HTTP/1.1 200 OK\nContent-Type: text/plain\n\nAcknowledged"

conn.send(response.encode("utf-8"))

conn.close()

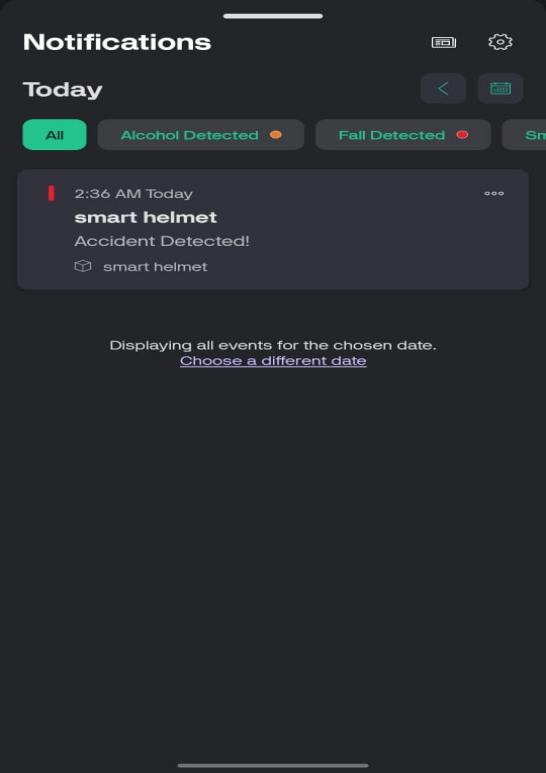
# Main script

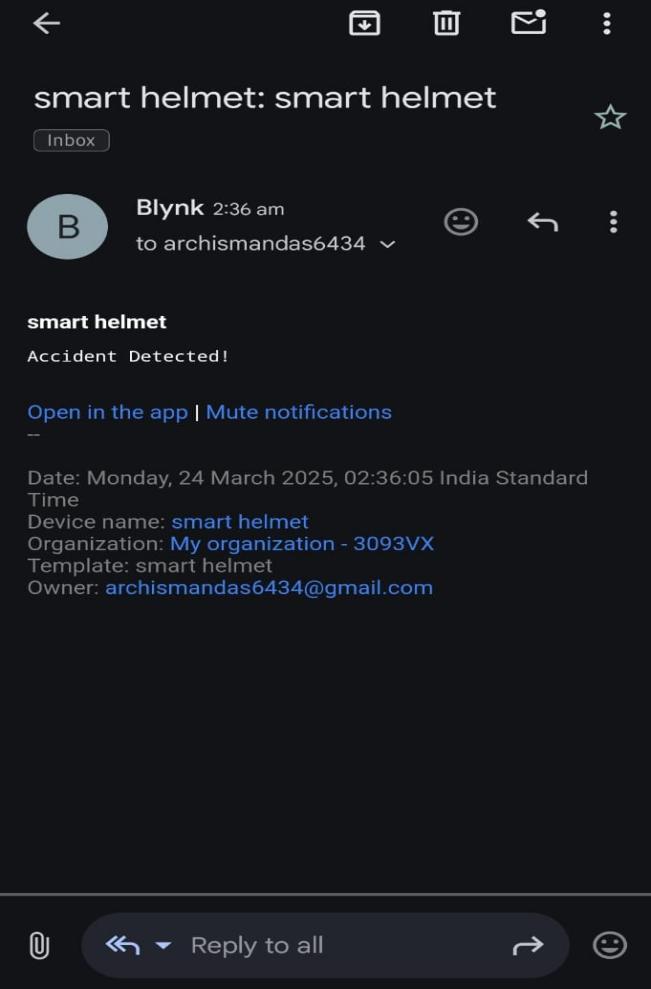
connect\_to\_wifi()

start\_server()

code for bike part

## Observation:



****

## Conclusion

Smart helmets represent a significant advancement in protective gear, combining safety with modern technology to enhance user experience. By integrating features such as impact detection, GPS navigation, hands-free communication, and health monitoring, these helmets go beyond basic protection, offering real-time assistance and increased situational awareness. Their applications extend across various fields, including motorcycling, construction, and military operations, making them a versatile and essential innovation. As technology continues to evolve, smart helmets are expected to become even more sophisticated, contributing to a safer and more connected world.