
Figure 1: .

# EC LAB REPORT

*Course: EE381, EC*

**Nikhil Jain** Roll No: 220709

**Khushboo Kumari** Roll No: 220527

Dept: Electrical Engineering

Date of submission: 20 April 2025

Section: B8 (Wednesday)

Table No: 20

# LAB WRITE UP

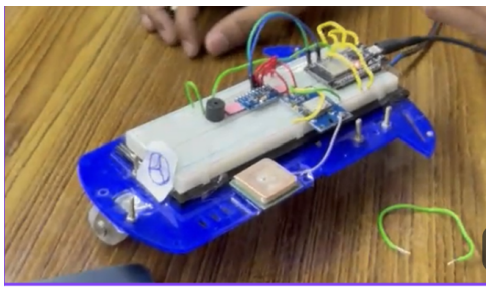

Figure 2: Final image of Project

**Q1: What problem are you trying to solve, and why is it important/interesting?**

We are building a real-time crash detection and alert system using an ESP32 module, an MPU6050 accelerometer, a NEO-6M GPS module, and a buzzer. The system detects abnormal vehicle forces (crashes), locks GPS location, and immediately sends an emergency message via Telegram. This is important because faster crash detection and location sharing can significantly improve emergency response time and save lives.

**Q2: What are list the existing solutions? any shortcomings in Describe a few of them. Is your them and solution approach unique in some way?**

Existing solutions include smartphone-based crash detection apps (like Google Pixel's Car Crash Detection, OnStar Guardian) and in-car systems (like Tesla, GM's OnStar, and BMW Assist).

**Shortcomings:**

- Smartphone apps rely on users carrying the phone correctly and keeping location/services active.
- In-built car systems are expensive, closed-source, and mostly available only in high-end vehicles.
- Both can have delays or failures in poor network/GPS conditions.

Our approach is unique because it's low-cost, independent of smartphones, uses dedicated sensors (accelerometer + GPS), and can be retrofitted into any vehicle — making crash detection accessible for everyone, especially in developing regions.

**Q3: What resources do you require to complete the project? Give a breakdown of the tasks that you need to accomplish week by week to complete the project.**

**Resources Required:**

- ESP 32
- MPU6050
- NEO-6M GPS

- Buzzer
- Car Model
- Jumper wires, Breadboard
- Arduino IDE software

**Weekly Task Breakup:**

- **Week 1:**
  - Learn about basic components: microcontroller, Bluetooth module, LCD, sensors.
  - Install Arduino IDE and Bluetooth Terminal App.
- **Week 2:**
  - Interface Bluetooth module with Arduino and test basic communication.
  - Interface sensor with Arduino and read sensor data.
  - Display sensor readings on the mobile Bluetooth terminal.
  - Worked on the Arduino Code.
- **Week 3:**
  - Integrate everything: sensor input, Bluetooth communication, and LCD display.
  - Implement logic for controlling outputs (e.g., buzzer/ignition based on sensor value).
  - Test and debug the complete setup.
  - Finalize the project: organize the circuit neatly and prepare for demonstration.

# LAB REPORT

## PROJECT OBJECTIVE:

The primary objective of this project is to design and implement a smart, low-cost crash detection and alert system using an ESP32 microcontroller, an MPU6050 accelerometer-gyroscope module, a NEO-6M GPS module, and a buzzer. The system aims to detect sudden impacts or crashes based on acceleration data, obtain the GPS location of the crash site, and immediately send an alert to a predefined user through Telegram, including the precise location of the incident.

This system is particularly useful in remote areas or highway scenarios where timely human intervention may not be possible. By automating crash alerts, the project addresses the delay in accident reporting and the need for real-time location sharing, potentially saving lives and enabling quicker emergency responses.

## RESOURCES USED:

**Hardware Components:**

- ESP32 WROOM 32 Development Board
- MPU6050 Accelerometer and Gyroscope Module
- NEO-6M GPS Module
- Piezoelectric Buzzer
- Car model chassis for mounting hardware (for demonstration/testing)
- Jumper Wires (Male-to-Male, Male-to-Female)
- Breadboard (for prototyping circuit connections)

**Software:**

- Arduino IDE (for coding and uploading firmware)
- Tiny GPS++ Arduino library
- Wi-Fi and HTTP Client Arduino libraries
- Telegram Bot API

## COST ANALYSIS:

| Component | Quantity | Unit Cost (INR) | Total Cost (INR) |
|---|---|---|---|
| ESP32 WROOM 32 Board | 1 | 400 | 400 |
| MPU6050 Module | 1 | 150 | 150 |
| NEO-6M GPS Module | 1 | 350 | 350 |
| Buzzer | 1 | 20 | 20 |
| Car Model | 1 | 200 | 200 |
| Jumper Wires & Breadboard | 1 set | 100 | 100 |
| **Total Estimated Cost** | | | **1,220** |

# COMPONENT DESCRIPTION & PIN CONNECTIONS:

### 0.1 ESP32 WROOM 32

The ESP32 is a powerful dual-core microcontroller with built-in Wi-Fi and Bluetooth capabilities, ideal for IoT applications. In this project, it serves as the central processing unit that collects sensor data, handles crash detection logic, interfaces with GPS and buzzer, and sends Telegram alerts over Wi-Fi.

We had to download a driver for the COM Port to be detected.

### 0.2 MPU6050 Accelerometer + Gyroscope Module

The MPU6050 is a 6-axis motion tracking device that measures acceleration and rotation. It communicates with the ESP32 via $I^2C$ protocol.

**Pin Connections (MPU6050 to ESP32):**

| MPU6050 Pin | ESP32 Pin |
|---|---|
| VCC | 3.3V |
| GND | GND |
| SDA | GPIO21 |
| SCL | GPIO22 |

### 0.3 NEO-6M GPS Module

This module provides GPS data including latitude and longitude. It communicates with the ESP32 using UART protocol.

**Pin Connections (NEO-6M to ESP32):**

| GPS Pin | ESP32 Pin |
|---|---|
| VCC | 3.3V |
| GND | GND |
| TX | GPIO16 |
| RX | GPIO17 |

## 0.4 Buzzer

The buzzer is used to provide an audible alert when a crash is detected.

**Pin Connections (Buzzer to ESP32):**

| Buzzer Pin | ESP32 Pin |
|---|---|
| + | GPIO4 |
| – | GND |

## Circuit Diagram

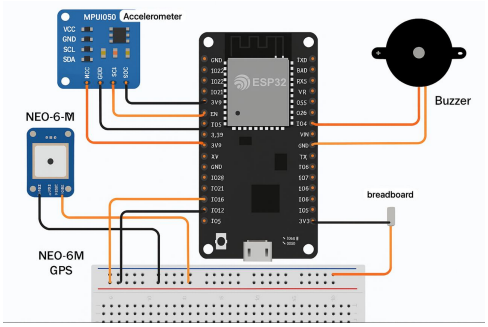


Figure 3: A visual representation

## IDE SETUP

CP210x Universal Windows Driver was downloaded on mac from `https://www.silabs.com/developers/usb-to-uart-bridge-vcp-drivers` for the COM port to be detected.

On IDE,

- ESP32 Dev Module was selected on the boards
- Old bootloader was selected.
- Additional Board Manager URLs: We had to paste `https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json`
- 115200 Baud rate was set on serial monitor while running

## Code

Listing 1: Crash Detection Code

```
#include <Wire.h>
#include <MPU6050.h>
#include <WiFi.h>
#include <HTTPClient.h>
#include <TinyGPS++.h>
#include <HardwareSerial.h>

// WiFi
```

```
const char* ssid = "iphone13";
const char* password = "nikhils23j";

// Telegram
String botToken = "8084528112:AAGIYHP5aF9EZV2S-4Z9y9jj-2M5z8gNLe0";
String chatID = "1519496749";

// Buzzer
#define BUZZER_PIN 4

// Accelerometer
MPU6050 accel;
#define SAMPLE_COUNT 200
#define CRASH_THRESHOLD 2.0
float offset_ax = 0, offset_ay = 0, offset_az = 0;

// GPS
#define GPS_RX 16
#define GPS_TX 17
TinyGPSPlus gps;
HardwareSerial gpsSerial(2);

// --- SETUP ---
void setup() {
  Serial.begin(115200);
  delay(1000);

  // Buzzer and I2C Init
  pinMode(BUZZER_PIN, OUTPUT);
  Wire.begin(21, 22);  // SDA, SCL

  // WiFi Connect
  Serial.println("Connecting to WiFi...");
  WiFi.begin(ssid, password);
  int retries = 0;
  while (WiFi.status() != WL_CONNECTED && retries < 20) {
    delay(500);
    Serial.print(".");
    retries++;
  }
  if (WiFi.status() == WL_CONNECTED) {
    Serial.println("\nWiFi connected.");
    sendTelegramMessage("       ESP32 Connected to WiFi.");
  } else {
    Serial.println("\nFailed to connect to WiFi.");
  }

```

```
  // Accelerometer Init & Calibration
  accel.initialize();
  Serial.println("Calibrating accelerometer... Please keep still.");

  long sum_ax = 0, sum_ay = 0, sum_az = 0;
  int16_t ax, ay, az;
  for (int i = 0; i < SAMPLE_COUNT; i++) {
    accel.getAcceleration(&ax, &ay, &az);
    sum_ax += ax;
    sum_ay += ay;
    sum_az += az;
    delay(50);
  }
  offset_ax = sum_ax / (float)SAMPLE_COUNT;
  offset_ay = sum_ay / (float)SAMPLE_COUNT;
  offset_az = sum_az / (float)SAMPLE_COUNT;
  Serial.println("Accelerometer calibrated.");

  // GPS Setup
  gpsSerial.begin(9600, SERIAL_8N1, GPS_RX, GPS_TX);
  Serial.println("Waiting for GPS fix...");
  unsigned long startTime = millis();
  bool fixAcquired = false;
  while (!fixAcquired && millis() - startTime < 60000) {
    while (gpsSerial.available()) gps.encode(gpsSerial.read());
    if (gps.location.isValid()) {
      fixAcquired = true;
    } else {
      Serial.print(".");
      delay(500);
    }
  }
  if (fixAcquired) {
    Serial.println("\nGPS fix acquired!");
  } else {
    Serial.println("\nGPS fix NOT acquired.");
  }
}

// --- LOOP ---
void loop() {
  // Read Accel
  int16_t ax_raw, ay_raw, az_raw;
  accel.getAcceleration(&ax_raw, &ay_raw, &az_raw);

  float ax = (ax_raw - offset_ax) / 16384.0;
```

```
  float ay = (ay_raw - offset_ay) / 16384.0;
  float az = (az_raw - offset_az) / 16384.0;

  float g_total = sqrt(ax * ax + ay * ay + az * az);
  Serial.print("g_total: "); Serial.println(g_total);

  // Feed GPS
  while (gpsSerial.available()) gps.encode(gpsSerial.read());

  // Detect Crash
  if (g_total > CRASH_THRESHOLD) {
    digitalWrite(BUZZER_PIN, HIGH);
    delay(1000);
    digitalWrite(BUZZER_PIN, LOW);

    String message;
    if (gps.location.isUpdated()) {
      String lat = String(gps.location.lat(), 6);
      String lng = String(gps.location.lng(), 6);
      message = "       Crash Detected! \ n     Location:
         https://maps.google.com/?q=" + lat + "," + lng;
    } else {
      message = "       Crash Detected! \ n     Location not
          available.";
    }

    sendTelegramMessage(message);
    delay(3000); // Avoid spamming
  }

  delay(100);  // 10 Hz
}

// --- TELEGRAM ---
void sendTelegramMessage(String message) {
  String safeMessage = urlencode(message);
  String url = "https://api.telegram.org/bot" + botToken +
               "/sendMessage?chat_id=" + chatID + "&text=" +
                  safeMessage;

  HTTPClient http;
  http.begin(url);
  int httpCode = http.GET();
  if (httpCode > 0) {
    Serial.println("Message sent! HTTP code: " + String(
       httpCode));
  } else {
```

```
    Serial.println("Failed to send message. HTTP error: " +
        String(httpCode));
  }
  http.end();
}

String urlencode(String str) {
  String encoded = "";
  char c;
  char code0, code1;
  for (int i = 0; i < str.length(); i++) {
    c = str.charAt(i);
    if (isalnum(c)) {
      encoded += c;
    } else {
      code1 = (c & 0xf) + '0';
      if ((c & 0xf) > 9) code1 = (c & 0xf) - 10 + 'A';
      c = (c >> 4) & 0xf;
      code0 = c + '0';
      if (c > 9) code0 = c - 10 + 'A';
      encoded += '%';
      encoded += code0;
      encoded += code1;
    }
  }
  return encoded;
}
```

## CODE EXPLANATION:

The system's logic is implemented in Arduino C++ using several libraries for sensor communication, Wi-Fi, and Telegram API handling. Below is a breakdown of the core code components and their functionalities:

**Included Libraries**

- `#include <Wire.h>`
  Enables $I^2C$ communication, used by the MPU6050 sensor.
- `#include <MPU6050.h>`
  A library to interact with the MPU6050 sensor for acceleration readings.
- `#include <WiFi.h>` and `#include <HTTPClient.h>`
  For ESP32's WiFi connection and making HTTP requests (used for Telegram messaging).
- `#include <TinyGPS++.h>` and `#include <HardwareSerial.h>`
  Handle GPS data parsing from the NEO-6M module using a second hardware serial port.

**Wi-Fi Setup**

The ESP32 connects to the local WiFi using:

```
WiFi.begin(ssid, password);
```

It attempts connection with a retry loop. Upon successful connection, it sends a confirmation message to Telegram using the `sendTelegramMessage()` function.

**Accelerometer Calibration (offset removal)**

The accelerometer is calibrated at startup:

- It takes 200 readings while the system is kept stationary.
- Averages of X, Y, and Z acceleration are calculated to get offset values (ax_offset, etc.).
- These offsets are subtracted from future readings to improve accuracy.

This step ensures the system correctly identifies abnormal motion (i.e., crashes) rather than reacting to minor vibrations or tilts.

The offset values were changing drastically with each reconnection. We cannot store a global offset values by single calculation, hence a sleep period of 30–45 sec was necessary.

**GPS Setup and Fix Check**

The GPS module is initialized over hardware serial (UART pins 16 and 17). The GPS module needs time to lock in with the satellite. Hence, the system enters a loop for up to 60 seconds, waiting for a valid GPS location:

```
if (gps.location.isValid())
```

→ This confirms a satellite fix, critical for reliable crash location reporting.

If the fix is not obtained in that time, the system still continues but marks the GPS location as "not available."

**Main Loop Logic**

This continuously runs after setup:

- Acceleration is read from MPU6050.
- Offset is subtracted and converted to g-force (using 16384.0 LSB/g scale).
- The total g-force is calculated using:

$$g_{\text{total}} = \sqrt{a_x^2 + a_y^2 + a_z^2}$$

**Crash Detection Logic:**

```
if (g_total > CRASH_THRESHOLD)
```

→ A crash is considered if `g_total` exceeds a set threshold (default = 2.0 g). This can be fine-tuned based on real driving data.

Currently, the threshold is set at 2g. Since we're moving in just one direction, it's difficult to get high jerk values, and noticeable jerks are only felt around 2g. This threshold was chosen based on numerous testing trials, manually seeing what could be the optimal threshold.

If a crash is detected:

- The buzzer is activated for 10 seconds.
- The GPS module is checked for updated coordinates.
- A Google Maps URL is constructed using the latitude and longitude.
- The crash alert is sent via Telegram.
- The `delay(3000)` ensures that repeated crash alerts aren't spammed.

**Telegram Message Sending**
The `sendTelegramMessage()` function creates a URL-encoded HTTP GET request to the Telegram Bot API. It includes:

- Bot Token
- Chat ID (of the recipient)
- Encoded message with location and crash alert

**URL Encoding Function**
The `urlencode()` function escapes special characters in the message string to ensure they're transmitted correctly via the HTTP GET request.

## TESTING & RESULTS

Testing was done by simulating different motion levels using the car model. Results were as follows:

- Normal vibrations and tilts: No false positives observed after calibration.
- Simulated crash (hard slap or sudden jerk): System detected g_total ¿ 2.0 and triggered buzzer + Telegram alert.
- GPS location was successfully fetched in most cases, though indoors it was slower to fix.
- Telegram message received within 3–5 seconds of detection.
- Buzzer was played for 10 seconds to alert nearby people and gather necessary medical attention.

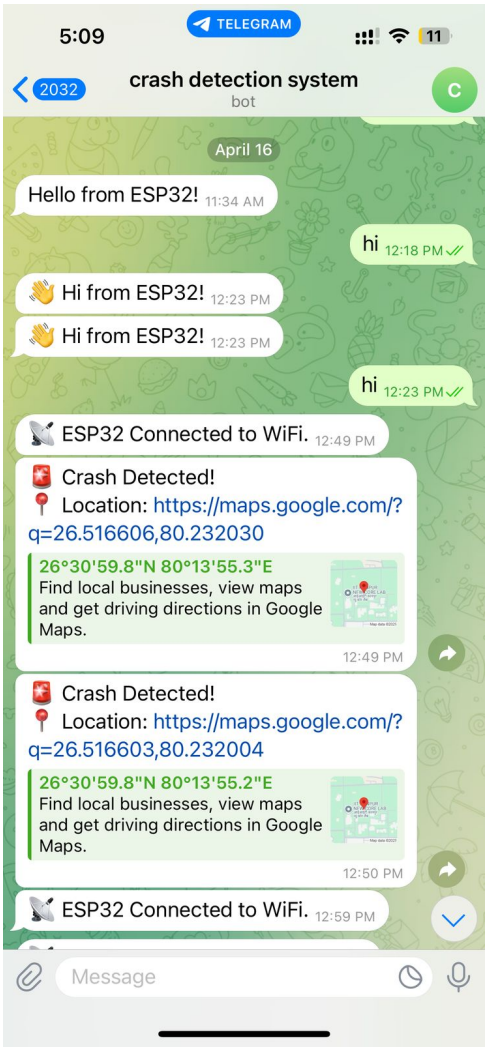


Figure 4: Notification on the Bot via WIFI

**Results Summary:**

- Crash detection was accurate and sensitive.
- GPS worked reliably outdoors.
- Telegram alerts were consistently delivered.
- System consumes low power and is compact enough for real vehicle applications.

## CONCLUSION:

This project demonstrates a compact, real-time accident detection and alert system using affordable electronics. The combination of acceleration sensing and GPS tracking, integrated with Telegram alerts, provides a highly effective tool for road safety. The system can be further enhanced by:

- Adding a battery and GSM module for complete portability
- Using an OLED screen to display status and medical information of the person driving the car

- Uploading crash logs to the cloud for analysis

The low cost, high reliability, and ease of setup make this solution viable for applications in cars, bikes, and even helmet-based safety systems.