

IOT-BASED AUTOMATED ROAD ACCIDENT **EMERGENCY CARE SYSTEM**

Submitted in partial fulfillment of the requirements for the degree of

Bachelor of Technology **in** **ECE** **Electronics and communication engineering**

by

SAKKTHI PRAANESH S M
20BEC0761

Under the guidance of

Prof. / Dr.

MUGELAN

SENSE

VIT, Vellore.



VIT[®]
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

MAY, 2024

DECLARATION

I hereby declare that the thesis entitled “IOT-BASED AUTOMATED ROAD ACCIDENT EMERGENCY CARE SYSTEM” submitted by me, for the award of the degree of **Bachelor of Technology in ECE** to VIT is a record of bonafide work carried out by me under the supervision of **Dr . MUGELAN R K**

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place : Vellore

Date: 30/04/2024



Signature of the Candidate

CERTIFICATE

This is to certify that the thesis entitled “IOT-BASED AUTOMATED ROAD ACCIDENT EMERGENCY CARE SYSTEM” submitted by **SAKKTHI PRAANESH S.M & 20BEC0761, SENSE**, VIT, for the award of the degree of **Bachelor of Technology in ECE**, is a record of bonafide work carried out by him / her under my supervision during the period, 03. 01. 2024 to 30.04.2024, as per the VIT code of academic and research ethics.

The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university. The thesis fulfills the requirements and regulations of the University and in my opinion meets the necessary standards for submission.

Place : Vellore

Date : 30/04/2024

Signature of the Guide

Internal Examiner

External Examiner

Head of the Department

ECE

ACKNOWLEDGEMENTS

With immense pleasure and deep sense of gratitude, we wish to express my sincere thanks to my supervisor **Dr. MUGELAN R K**, Associate Professor Senior, SENSE, VIT University, without her motivation and continuous encouragement, this project work would not have been successfully completed.

We are grateful to the Chancellor of VIT University, **Dr. G. Viswanathan**, the Vice Presidents, the Vice Chancellor for motivation us to carry out research in the VIT University and also for providing me with infrastructural facilities and many other resources needed for my project.

We express our sincere thanks to **Dr. Sivanantham S**, Dean, SENSE, VIT University and **Dr. Noor Mohammed V**, HOD, SENSE, VIT University for their kind words of support and encouragement. We like to acknowledge the support rendered by our colleagues in several way throughout our project work.

PLACE : Vellore

DATE : 30/04/2024



Sakthi Praanesh S M

Executive Summary

The project endeavors to mitigate preventable fatalities resulting from accidents by developing a system that integrates ultrasonic sensor, pressure sensor, and GPS technology. This system is intended to promptly alert emergency services following an accident occurrence.

Engineered to seamlessly integrate with vehicles through OEM, the system automatically dispatches an alert message to emergency services, furnishing details regarding the accident's location and severity. This swift notification enables emergency responders to take immediate action.

By furnishing timely information to emergency services, the system aspires to curtail the number of preventable fatalities, ultimately saving lives. The incorporation of sensors and GPS technology underscores the potential of IoT in enhancing safety measures and mitigating the impact of accidents.

	CONTENTS	Page No.
	Acknowledgement	4
	Executive Summary	5
	Table of Contents	6
	List of Figures	7
	List of Tables	8
	Abbreviations	9
1	INTRODUCTION	11
	1.1 OBJECTIVE	11
	1.2 MOTIVATION	12
	1.3 BACKGROUND	12
2	PROJECT DESCRIPTION AND GOALS	13
	2.1. PROPOSED SYSTEM	13
	2.2. EXSISTING SYSTEM COMPARISION	14
	2.3.LITREATURE REVIEW	14
3	TECHNICAL SPECIFICATION	16
	3.1. HARDWARE AND SOFTWARE SPECIFICATIONS	16
4	DESIGN APPROACH AND DETAILS	18.
	4.1 DESIGN APPROACH	18.
	4.2 CODES AND STANDARDS	20.
	4.3 CONSTRAINTS, ALTERNATIVES AND TRADEOFFS	41
5	SCHEDULE, TASKS AND MILESTONES	43.

6	RESULT & DISCUSSION	44
		.
	6.1 HARDWARE OUTPUTS	44
	6.2 SOFTWARE OUTPUTS	49
	6.3 PROJECT DEMONSTRATION	51
7	SOCIAL IMPACT	53
8	SUMMARY	53
9	REFERENCES	54

List of Figures

Figure No.	Title	Page No.
1.1	Emergency alert system process	11
2.1	Layout of working mechanism	13
3.1	Tools used in hardware part	17
3.2	Tools used in software part	17
4.1	Circuit diagram	18
4.2	Data flow diagram	19
4.3	Use case diagram	19
4.4	Ultrasonic sensor	20
4.5	GPS sensor	22
4.6	GSM module	24
4.7	Pressure sensor	27
4.8	Login page of web	35
6.1	Ultrasonic sensor implementation	44
6.2	Ultrasonic sensor output	45
6.3	GPS sensor implementation	45
6.4	GPS sensor output	46
6.5	Pressure sensor implementation	47
6.6	Pressure sensor output	47
6.7	GSM module implementation	48
6.8	Frontend responsive page	49
6.9	Cloud firestore	49
6.10	Intermediate redirector	50
6.11	Mobile SMS	50
6.12	API endpoint	51
6.13	Prototype of car	51
6.14	Installation of components based on circuit	52

List of Tables

Table No.	Title	Page No.
5.1	Timeline	43

Abbreviations

OEM	Original Equipment Manufacturer
GPS	Global Positioning System
GSM	Global System for Mobile communication
FSR	Force sensing resistor
API	Application Programming Interface
CORS	Cross Origin Resource Sharing
DOM	Document Object Model
SVM	Support Vector Machine
ACA	Ant Colony algorithm

1. INTRODUCTION

1.1.OBJECTIVE

The system is filled with a GPS device that furnishes precise vehicle location information, seamlessly integrated with the vehicle via the Original Equipment Manufacturer (OEM). In case of an accident, the system employs GSM technology to dispatch an alert message containing real-time location data to both emergency services and the police station.

Additionally, Ultrasonic and Pressure sensors are deployed to ascertain accident occurrences. A predefined range is established, and if surpassed, the vehicle's pressure, facilitated by OEM connectivity, is gauged to discern the accident severity. Should the accident be deemed severe, necessitating immediate medical intervention, the system dispatches an alert message to the requisite emergency services.

Utilizing GPS technology, the system continuously tracks the vehicle's whereabouts. In the event of an accident, an alert message featuring the live location is dispatched to emergency services and the police station via GSM technology. Furthermore, ultrasonic sensors and pressure sensors are incorporated to gauge accident severity. Upon surpassing a predetermined pressure threshold indicative of a major accident, the system promptly issues an emergency alert to the pertinent services.

Coupled with a full-stack app, this comprehensive system enables swift response and provides pertinent accident details.



Fig.no 1.1
Emergency alert system process

1.2.MOTIVATION

The project overview entails a system designed to detect accident severity and promptly notify the emergency service center. In instances of significant accidents requiring urgent attention, the system will immediately transmit a direct notification along with the location. Several methods have been employed for accident detection, including pressure sensors, ultrasonic sensors, GPS, GSM, and machine learning algorithms. While literature discusses various strategies for accident detection and prevention, a comprehensive survey is notably absent. This paper aims to address this gap by critically reviewing literature on accident detection and reporting systems. The objective is to provide a comprehensive understanding of existing techniques to develop effective systems that leverage strengths and overcome challenges encountered in current systems..

1.3.BACKGROUND

The Design Project holds significance within the organization's operations as it strives to guarantee the secure commuting of employees to and from the workplace via their vehicles. Employee safety stands as a paramount concern for the employer, and this system endeavors to tackle the potential occurrence of accidents. Although some accidents may be beyond prevention, timely access to accurate accident data enables swift intervention, potentially averting loss of life.

2. PROJECT DESCRIPTION AND GOALS

2.1.PROPOSED SYSTEM

Automatic accident detection systems play a crucial role in reducing casualties and damages resulting from road accidents. The proposed system aims to enhance the precision and efficacy of accident detection and prevention by addressing the challenges faced by current systems.

The paper conducts a literature review on techniques for accident detection and prevention, shedding light on factors contributing to accidents and methodologies for prevention. The proposed system consists of three modules. The first module monitors ground clearance and detects any changes within a specified range, signaling the occurrence of an accident. The second module utilizes a pressure sensor to identify impacts and activates the GSM module to send an alert message to emergency services. The third module assesses the severity of the accident, with the alert message containing the real-time location of the accident site.

The proposed system overcomes the limitations of gyroscope sensors, which rely on accelerometer sensors to measure tilt changes and GPS for speed measurement, triggering alerts upon accident detection. Gyroscope sensors may inaccurately detect tilt changes when a vehicle traverses slopes, leading to false alerts to emergency services. Therefore, the proposed system employs ultrasonic sensors to measure ground clearance, eliminating false alerts. Moreover, the threshold limit for the ultrasonic sensor is expanded to a certain range to prevent false alerts triggered by speed bumps and small obstacles encountered by the vehicle.



Fig.no 2.1
Layout of Working Mechanisim

2.2.EXSISTING SYSTEM COMPARISION

Numerous accident detection systems have been proposed previously. One example involves the utilization of two sensors: a MEMS sensor for angle detection and a vibration sensor to identify vehicle movement. Another system employs IoT and cloud computing, utilizing Support Vector Machine (SVM) with Ant Colony Algorithm (ACA) for vehicle detection. This system employs magneto resistive sensors to monitor vehicles. Its aim is to distinguish between accidents in traffic and those in non-traffic areas. Furthermore, the system offers accident location data using ATmega 328 Microcontroller and RF transmitter/receiver technology.

DRAWBACKS:

- This system is designed to detect accidents only in traffic conditions.
- The location accuracy is low.
- The system uses Atmega microcontroller.

2.3.LITREATURE REVIEW

[7]The system provides an idiosyncratic prevention and detection system that dispenses the ultimate panacea for drivers which ensures safety and prevents loss of life by taking appropriate measures in right time. It also checks whether the driver is drowsy or in an unstable state which can lead to pedal mix-up and in some cases unintended acceleration or turning of the steering wheel to the wrong direction which can lead to crashing of the vehicle with other vehicles or concrete road barrier.

[6]The proposed approach aims to take advantage of advanced specifications of smartphones to design and develop a low-cost solution for enhanced transportation systems that is deployable in legacy vehicles. In this context, a customized Android application is developed to gather information regarding speed, gravitational force, pressure, sound, and location. The speed is a factor that is used to help improve the

identification of accidents. It arises because of clear differences in environmental conditions that arise in low speed collisions, versus higher speed collisions. The information acquired is further processed to detect road incidents.

[10] It proposes to utilize the capability of a GPS receiver to monitor speed of a vehicle and detect accident basing on monitored speed and send accident location to an Alert Service Center. The GPS will monitor speed of a vehicle and compare with the previous speed in every second through a Microcontroller Unit. Whenever the speed will be below the specified speed, it will assume that an accident has occurred. The system will then send the accident location acquired from the GPS along with the time and the speed by utilizing the GSM network

[2] This paper proposes the model based on Internet of Things and Cloud Connectivity which can detect accident of the vehicle and is able to notify the nearby emergency services about the accident.

[3] This paper critically examines existing methodologies for predicting and preventing road accidents, highlighting strengths, limitations, and challenges with escalating fatalities due to traffic hazards

[1] This paper presents a novel framework for automatic road accident detection using deep learning techniques, significantly improving accuracy and reducing false positives, by integrating multi-modal sensor data and leveraging advanced convolutional neural networks.

[5] Aim of this project is to use IoT to detect the Car Accident and alert the required authority with intensity of accident for emergency cases and also measure the car health with specific parameters. Every device is connected to each other in IOT domain. IOT can be used to solve the current problems by getting the sensor data and detection and intimation about the accidents.

[9] The accident detection system communicates the accelerometer values to the processor which continuously monitors for erratic variations. When an accident occurs, the related details are sent to the emergency contacts by utilizing a cloud based service. The vehicle location is obtained by making use of the global positioning system. The system promises a reliable and quick delivery of information relating to the accident in real time and is appropriately named Konnect.

.

3. TECHNICAL SPECIFICATION

3.1.HARDWARE AND SOFTWARE SPECIFICATIONS

The initial phase of the requirements analysis process involves compiling the specifications for a specific software system, encompassing functional, performance, and security requisites. These requirements also outline usage scenarios from user, operational, and administrative viewpoints. The Software Requirements Specification (SRS) serves to offer a comprehensive outline of the software project, encompassing its objectives and parameters. It delineates the project's intended audience, as well as its user interface, hardware, and software prerequisites. Additionally, the SRS articulates how the client, project team, and end-users perceive the project and its functionalities.

HARDWARE REQUIREMENTS AND INTERFACE:

1. Arduino UNO Board
2. Ultrasonic Sensor
3. GSM Module
4. GPS Module
5. Force Pressure Sensor
6. 10k Ohm Resistor
7. Jumper cables
8. 220 Ohm resistor
9. Bread Board
- 10.LED light

\

Internal connections utilize Wi-Fi, while external connections are facilitated through the utilization of an Arduino UNO board hardware device.

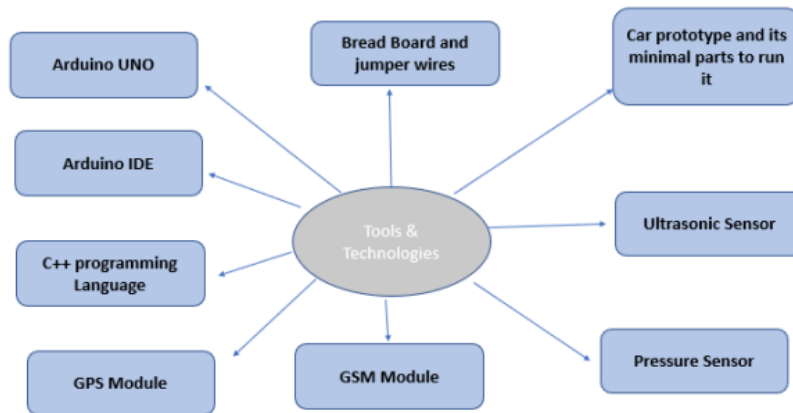


Fig.no 3.1
Tools used in hardware part

SOFTWARE REQUIREMENTS AND SPECIFICATIONS:

- 1) Using IoT - IDE : Arduino
- 2) React JS
- 3) Firebase DB
- 4) VS code
- 5) JS,HTML,CSS

Arduino IDE software is employed for programming. Additionally, the server and front-end interface are accessed via console and webpage, respectively.

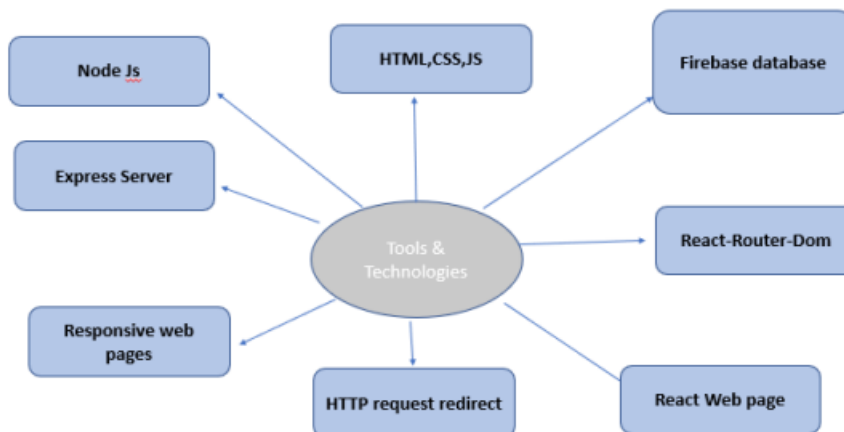


Fig.no 3.2
Tools used in software part

4. DESIGN APPROACH AND DETAILS

4.1.DESIGN APPROACH

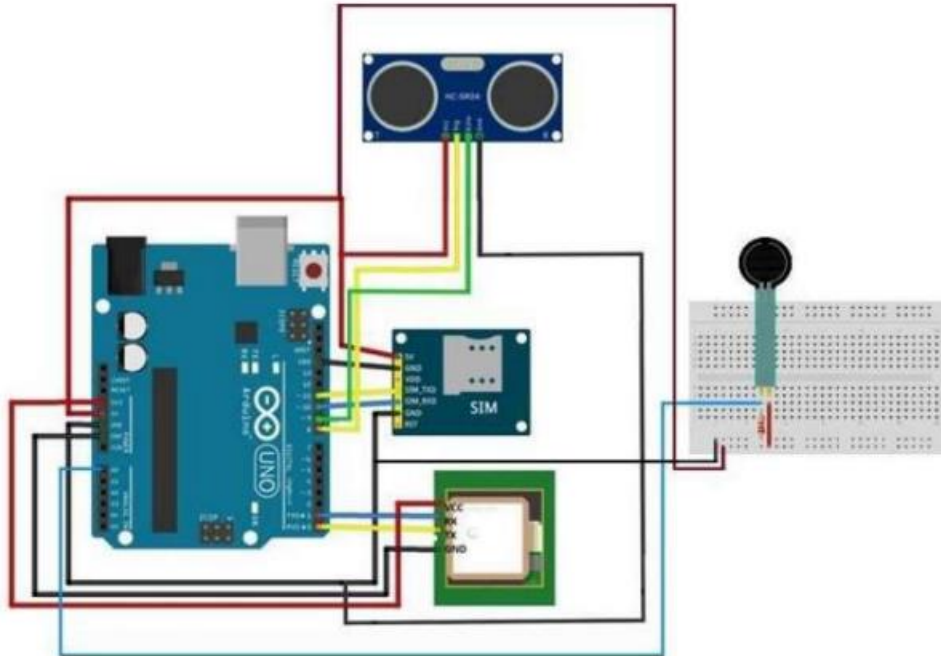


Fig.no 4.1
Circuit diagram

As said before the proposed consists of three modules

First module:

It checks the ground clearance, and if any changes occur within the specified range, it detects that an accident has occurred.

Second module:

Here it uses a pressure sensor to detect the impact and triggers the GSM module to send an alert message to the emergency number.

Third module:

It determines the seriousness of the accident and the alert message sent which includes the live locations of the accident

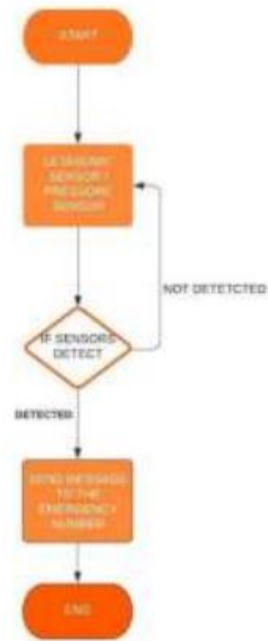


Fig.no 4.2
DATAFLOW DIAGRAM

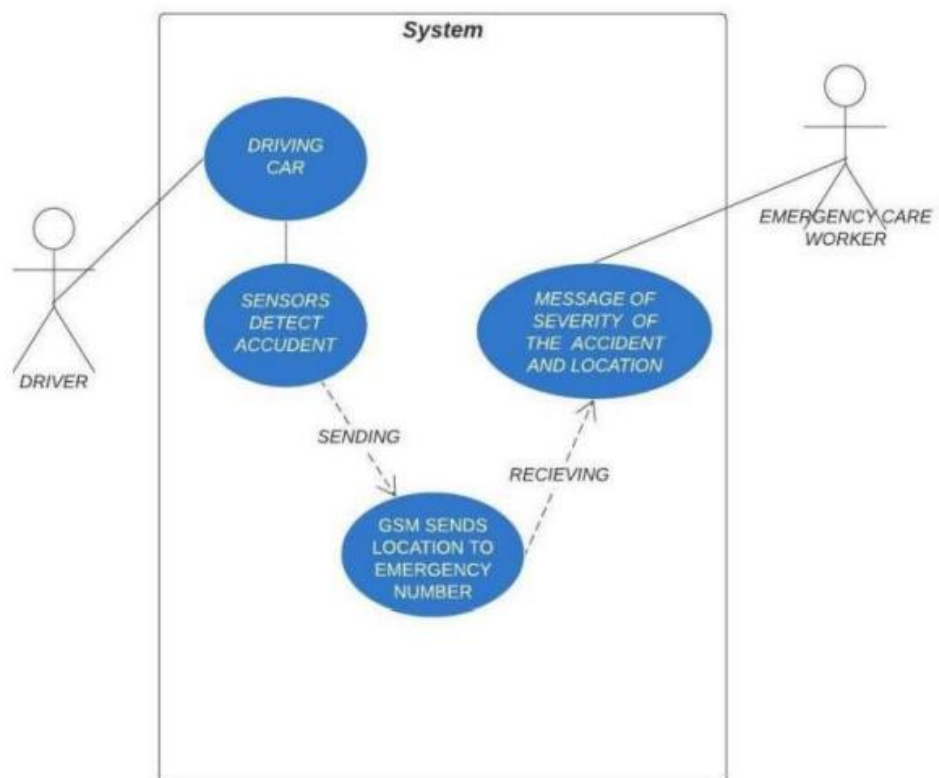


Fig.no 4.3
USECASE DIAGRAM

4.2.CODES AND STANDARDS

HARDWARE PARTS:

ULTRASONIC SENSOR TEST:



Fig.no 4.4
ULTRASONIC SESNSOR

```
const int trigPin = 3;
const int echoPin = 2;
// defines variables
long duration;
int distance;
void setup() {
  pinMode(trigPin, OUTPUT); // Sets the trigPin as an Output
  pinMode(echoPin, INPUT); // Sets the echoPin as an Input
  Serial.begin(9600); // Starts the serial communication
}
void loop() {
  // Clears the trigPin
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  // Sets the trigPin on HIGH state for 10 micro seconds
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  // Reads the echoPin, returns the sound wave travel time in
  microseconds
  duration = pulseIn(echoPin, HIGH);
  // Calculating the distance
  distance = duration * 0.034 / 2;
  // Prints the distance on the Serial Monitor
  Serial.print("Distance: ");
  Serial.println(distance);
}
```

The above code continuously measures distances using an ultrasonic sensor and outputs the results via the Serial Monitor for monitoring or debugging purposes

Functions used:

Setup Function: This function runs once when the Arduino board is powered on or reset, like `pinMode()` is used to set `trigPin` as an output and `echoPin` as an input, indicating their respective functionalities and also `Serial.begin()` initializes serial communication with a baud rate of 9600, enabling communication with the Arduino IDE Serial Monitor. (The setup function is been same for all embedded c codes which has been used among all the hardware parts)

Loop Function: This function runs continuously after the `setup()` function finishes

- `digitalWrite(trigPin, LOW)` - clears the trigger pin and reset the pin to low state
- `delayMicroseconds(2)` - introduces a small delay.
- `digitalWrite(trigPin, HIGH)` - sets the trigger pin to a high state for 10 microseconds to send an ultrasonic pulse.
- `delayMicroseconds(10)` - maintains the high state for a brief period.
- `Serial.print()` and `Serial.println()` - functions print the measured distance to the Serial Monitor for debugging or monitoring purposes.

GPS TEST:



Fig.no 4.5
GPS SENSOR

```
#include <TinyGPS++.h>
#include <SoftwareSerial.h>
/* Create object named bt of the class SoftwareSerial */
SoftwareSerial GPS_SoftSerial(6, 5); /* (Rx, Tx) */
/* Create an object named gps of the class TinyGPSPlus */
TinyGPSPlus gps;

volatile float minutes, seconds;
volatile int degree, secs, mins;

void setup() {
  Serial.begin(9600); /* Define baud rate for serial communication */
  GPS_SoftSerial.begin(9600); /* Define baud rate for software serial
communication */
}

void loop() {
  smartDelay(1000); /* Generate precise delay of 1ms */
  unsigned long start;
  double lat_val, lng_val, alt_m_val;
  uint8_t hr_val, min_val, sec_val;
  bool loc_valid, alt_valid, time_valid;
  lat_val = gps.location.lat(); /* Get latitude data */
  loc_valid = gps.location.isValid(); /* Check if valid location
data is available */
  lng_val = gps.location.lng(); /* Get longitude data */
  alt_m_val = gps.altitude.meters(); /* Get altitude data in
meters */
  alt_valid = gps.altitude.isValid(); /* Check if valid altitude
data is available */
  hr_val = gps.time.hour(); /* Get hour */
  min_val = gps.time.minute(); /* Get minutes */
  sec_val = gps.time.second(); /* Get seconds */
  time_valid = gps.time.isValid(); /* Check if valid time data
is available */
  if (!loc_valid)
  {
    Serial.print("Latitude : ");
    Serial.println("*****");
    Serial.print("Longitude : ");
    Serial.println("*****");
  }
  else
  {
    DegMinSec(lat_val);
    Serial.print("Latitude in Decimal Degrees : ");
    Serial.println(lat_val, 6);
    Serial.print("Latitude in Degrees Minutes Seconds : ");
    Serial.print(degree);
    Serial.print("\t");
    Serial.print(mins);
    Serial.print("\t");
    Serial.println(secs);
    DegMinSec(lng_val); /* Convert the decimal degree value
into degrees minutes seconds form */
    Serial.print("Longitude in Decimal Degrees : ");
    Serial.println(lng_val, 6);
    Serial.print("Longitude in Degrees Minutes Seconds : ");
```

```

        Serial.print(degree);
        Serial.print("\t");
        Serial.print(mins);
        Serial.print("\t");
        Serial.println(secs);
    }
    if (!alt_valid)
    {
        Serial.print("Altitude : ");
        Serial.println("*****");
    }
    else
    {
        Serial.print("Altitude : ");
        Serial.println(alt_m_val, 6);
    }
    if (!time_valid)
    {
        Serial.print("Time : ");
        Serial.println("*****");
    }
    else
    {
        char time_string[32];
        sprintf(time_string, "Time : %02d/%02d/%02d \n", hr_val,
min_val, sec_val);
        Serial.print(time_string);
    }
}

static void smartDelay(unsigned long ms)
{
    unsigned long start = millis();
    do
    {
        while (GPS_SoftSerial.available()) /* Encode data read from GPS
while data is available on serial port */
            gps.encode(GPS_SoftSerial.read());
        /* Encode basically is used to parse the string received by the GPS and
to store it in a buffer so that information can be extracted from it */
    } while (millis() - start < ms);
}

void DegMinSec( double tot_val) /* Convert data in decimal
degrees into degrees minutes seconds form */
{
    degree = (int)tot_val;
    minutes = tot_val - degree;
    seconds = 60 * minutes;
    minutes = (int)seconds;
    mins = (int)minutes;
    seconds = seconds - minutes;
    seconds = 60 * seconds;
    secs = (int)seconds;
}

```

The above code continuously reads GPS (location) data, parses it, extracts relevant information, and outputs it to the Serial Monitor for monitoring or further processing.

Functions used:

- ``smartDelay(1000);``: Generates a precise delay of 1000 milliseconds (1 second) using the ``smartDelay()`` function.
- GPS Data Parsing:

``gps.encode(GPS_SoftSerial.read());``: Reads data from the software serial port and parses it using the ``encode()`` function of the ``TinyGPSPlus`` library. This function decodes GPS data packets and updates the internal GPS object (``gps``) with the

parsed information.

- GPS Data Extraction:

Extracts latitude, longitude, altitude, and time data from the GPS object (`gps`). Checks if the extracted data is valid using `isValid()` functions. Converts latitude and longitude values from decimal degrees to degrees, minutes, and seconds using the `DegMinSec()` function.

- Serial Output:- Prints latitude, longitude, altitude, and time data to the Serial Monitor. If data is invalid or not available, prints "*****" instead of the actual data.
- `DegMinSec()` Function: Converts decimal degree values into degrees, minutes, and seconds format
- `smartDelay()` Function: Implements a delay function with interrupt-driven GPS data parsing. This function allows the Arduino to continuously process GPS data while waiting for the delay to elapse.

GSM TEST:



Fig.no 4.6
GSM MODULE

```

#include <SoftwareSerial.h>
#include <TinyGPS.h>
SoftwareSerial mySerial(9, 10);
char msg;
char call;
const int pingPin = 3; // Trigger Pin of Ultrasonic Sensor
const int echoPin = 2; // Echo Pin of Ultrasonic Sensor
int fsrAnalogPin = 0; // FSR is connected to analog 0
int fsrReading; // the analog reading from the FSR resistor divider
float lat, lon; // create variable for latitude and longitude object
SoftwareSerial gpsSerial(5, 4); // rx, tx
TinyGPS gps; // create gps object
void setup() {
    mySerial.begin(9600);
    Serial.begin(9600); // Starting Serial Terminal
    Serial.println("The GPS Received Signal:");
    gpsSerial.begin(9600);
}
void loop() {
    while(gpsSerial.available()){ // check for gps data
        if(gps.encode(gpsSerial.read())) // encode gps data
        {
            gps.f_get_position(&lat, &lon); // get latitude and longitude
            Serial.print(lat);
            Serial.print(" ");
            Serial.print(lon);
            Serial.print(" ");
        }
        String latitude = String(lat, 6);
        String longitude = String(lon, 6);
        Serial.println(latitude+" "+longitude);
        delay(1000);
        long duration, inches, cm;
        unsigned long seconds=1000L;
        pinMode(pingPin, OUTPUT);
        digitalWrite(pingPin, LOW);
        delayMicroseconds(2);
        digitalWrite(pingPin, HIGH);
        delayMicroseconds(10);
        digitalWrite(pingPin, LOW);
        pinMode(echoPin, INPUT);
        duration = pulseIn(echoPin, HIGH);
        inches = microsecondsToInches(duration);
        cm = microsecondsToCentimeters(duration);
        fsrReading = analogRead(fsrAnalogPin);
        Serial.print("Analog reading = ");
        Serial.println(fsrReading);
        Serial.print(inches);
        Serial.print("in, ");
        Serial.print(cm);
        Serial.print("cm");
        Serial.println();

        if(cm > 10 || fsrReading > 300) { //---> include this when gps is available
            mySerial.println("AT+CMGF=1"); //Sets the GSM Module in Text Mode
            delay(1000); // Delay of 1000 milli seconds or 1 second

            mySerial.println("AT+CMGS=\""+918778845799+"\r"); // Replace x with mobile
            number
            delay(1000);
            //mySerial.println("EMETGENCY!!!! SEND AN AMBULANCE TO THE SHARED
            LOCATION "); // The SMS text you want to send

            mySerial.println("emergency");// The SMS text you want to send
            delay(100);
            mySerial.println((char)26); // ASCII code of CTRL+Z delay(1000);
            exit(0);
        }
        delay(seconds);
    }
    long microsecondsToInches(long microseconds) { return microseconds / 74 /
    2;
    }
    long microsecondsToCentimeters(long microseconds) { return microseconds /
    29 / 2;
    }
}

```


The above code continuously reads GPS data, measures distances using an ultrasonic sensor, monitors an FSR, and triggers an emergency alert if certain conditions are met, while also providing feedback via the Serial Monitor.

It can connect to the network and can send message or can make a http request using sim card

Functions used:

- ``while(gpsSerial.available()) { ... }``: Checks if there is data available from the GPS module via ``gpsSerial``.
- ``if(gps.encode(gpsSerial.read())) { ... }``: Reads and parses GPS data using the ``encode()`` function of the ``TinyGPS`` library. If valid GPS data is decoded, it extracts latitude and longitude using ``f_get_position()`` and prints them to the Serial Monitor.
- Converts latitude and longitude values to strings with 6 decimal places
- Prints the latitude and longitude to the Serial Monitor.
- Sends a trigger signal to the ultrasonic sensor and measures the duration of the echo pulse to calculate the distance to an object.
- Reads the analog input from the FSR (Force-Sensing Resistor) and prints the analog reading from the FSR, distance measured by the ultrasonic sensor in inches and centimeters, and a blank line to the Serial Monitor.
- Checks if the distance measured by the ultrasonic sensor is greater than 10 cm or the analog reading from the FSR is greater than 300
- If either condition is met, it sends an SMS message to a predefined phone number using the GSM module.
- Delays the loop execution for 1 second before repeating the process.

And other Helper functions:

- ``long microsecondsToInches(long microseconds) { ... }``: Converts the duration of the echo pulse (in microseconds) to distance in inches.
- ``long microsecondsToCentimeters(long microseconds) { ... }``: Converts the duration of the echo pulse (in microseconds) to distance in centimeters.

PRESSURE SENSOR TEST:



Fig.no 4.7
PRESSURE SENSOR

```

int fsrPin = 0;      // the FSR and 10K pulldown are connected to a0
int fsrReading;      // the analog reading from the FSR resistor divider

void setup(void) {
  Serial.begin(9600);
}

void loop(void) {
  fsrReading = analogRead(fsrPin);

  Serial.print("Analog reading = ");
  Serial.print(fsrReading);      // print the raw analog reading

  if (fsrReading < 10) {
    Serial.println(" - No pressure");
  } else if (fsrReading < 200) {
    Serial.println(" - Light touch");
  } else if (fsrReading < 500) {
    Serial.println(" - Light squeeze");
  } else if (fsrReading < 800) {
    Serial.println(" - Medium squeeze");
  } else {
    Serial.println(" - Big squeeze");
  }
  delay(1000);
}

```

The above code continuously reads the analog voltage from the FSR, classifies the pressure applied based on the reading, and prints the result to the Serial Monitor. This allows for real-time monitoring of pressure changes applied to the FSR.

Functions used:

Loop functions:

- ``fsrReading = analogRead(fsrPin);``: Reads the analog voltage from the FSR (Force-Sensing Resistor) using the ``analogRead()`` function. This value represents the pressure applied to the FSR.
- Based on the value of ``fsrReading``, the code classifies the pressure applied to the FSR into different categories:
- If ``fsrReading`` is less than 10, it prints "No pressure" to the Serial Monitor.

ARDUINO IDE MAIN ACTIVITY (SOURCE CODE):

```
#include <SoftwareSerial.h>;
#include <TinyGPS.h>;
SoftwareSerial mySerial(9, 10);
char msg;
char call;

const int pingPin = 7; // Trigger Pin of Ultrasonic Sensor
const int echoPin = 6; // Echo Pin of Ultrasonic Sensor
int fsrAnalogPin = 0; // FSR is connected to analog 0
int fsrReading; // the analog reading from the FSR resistor divider
float lat, lon; // create variable for latitude and longitude object
SoftwareSerial gpsSerial(3,4); // tx, rx

TinyGPS gps; // create gps object

void setup() {
  mySerial.begin(9600);
  Serial.begin(9600); // Starting Serial Terminal
  Serial.println("The GPS Received Signal:");
  gpsSerial.begin(9600);
}

void loop() {

  while(gpsSerial.available()){ // check for gps data
    if(gps.encode(gpsSerial.read())) // encode gps data
    {
      gps.f_get_position(&lat,&lon); // get latitude and longitude

      Serial.print(lat);
      Serial.print(" ");
      Serial.print(lon);
      Serial.print(" ");
    }
  }

  String latitude = String(lat,6);
  String longitude = String(lon,6);
  Serial.println(latitude+" "+longitude);
  delay(1000);

  long duration, inches, cm;
  unsigned long seconds=1000L;
  pinMode(pingPin, OUTPUT);
  digitalWrite(pingPin, LOW);
  delayMicroseconds(2);
  digitalWrite(pingPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(pingPin, LOW);
  pinMode(echoPin, INPUT);
  duration = pulseIn(echoPin, HIGH);
```

```

    inches = microsecondsToInches(duration);
    cm = microsecondsToCentimeters(duration);
    fsrReading = analogRead(fsrAnalogPin);
    Serial.print(" Analog reading = ");
    Serial.println(fsrReading);
    Serial.print(inches);
    Serial.print(" in, ");
    Serial.print(cm);
    Serial.print(" cm");
    Serial.println();
    if(cm > 10 || fsrReading > 100){
        mySerial.println("AT+CMGF=1"); //Sets the GSM Module in Text Mode
        delay(1000); // Delay of 1000 milli seconds or 1 second
        mySerial.println("AT+CMGS=\"" + 918825739212 + "\"\r"); // Replace x with
        mobile number
        delay(1000);
        mySerial.println("EMETGENCY!!!! SEND AN AMBULANCE TO THE
        SHARED LOCATION" latitude and " longitude "); // The SMS text you want
        to send
        delay(100);
        mySerial.println((char)26); // ASCII code of CTRL+Z
        delay(1000);
        exit(0);
    }
    delay(seconds);
}

```

```

long microsecondsToInches(long
microseconds) {return microseconds / 74
/ 2;
}

long microsecondsToCentimeters(long
microseconds) {return microseconds / 29
/ 2;
}

```

Here with the mentioned code the whole circuit works which is the combination of all the codes of the above mentioned sensor.

Where as in this code it used the functions and features which has been implemented each of the mentioned sensor.

SOFTWARE AND WEB APPLICATION:

Here we use the web application as the additional feature because sometime during crucial circumstances may the sensors and the implemented modules get damaged. So this applications which has been connected to the phone even the sensors get damaged the information of the accident will be passed as an alert message to the respective mobile phone.

FRONT END

Index

```
import React from "react";
import ReactDOM from "react-dom/client";
import "./index.css";
import App from "./App";

const root = ReactDOM.createRoot(document.getElementById("root"));

root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

This above code here sets up a React application by creating a root element, importing necessary components and styles, and rendering the main component (`App`) wrapped in `React.StrictMode` into the DOM(document object model). This is a standard approach for initializing a React application.

Functions used:

- `import React from "react";`: Imports the React library, which is necessary for creating and working with React components.
- `import ReactDOM from "react-dom/client";`: Imports the ReactDOM library, which provides methods for rendering React components into the DOM.
- `import "./index.css";`: Imports a CSS file named `index.css`, which likely contains styles for the application.
- `import App from "./App";`: Imports the main component of the application from the `App.js` file.
- `const root = ReactDOM.createRoot(document.getElementById("root"));`: Creates a root element using `ReactDOM.createRoot()`. This root element serves as the entry point for rendering React components into the DOM. It typically corresponds to a `<div>` element with the id of "root".
- `root.render(...)`: Renders the main component (`App`) into the root element.
- `<React.StrictMode>`: Wraps the `App` component with `React.StrictMode`. This enables additional checks and warnings for potential issues in the application during development. It helps identify common mistakes and encourages better practices.
- `<App />`: Renders the `App` component. This is the top-level component of the application, and it may contain other components nested within it.

App

```
import { BrowserRouter, Routes, Route } from "react-router-dom";
import Login from "../components/Login";
import Register from "../components/Register";
import Navbar from "../components/Navbar";
import Homeabout from "../components/Homeabout";
import Homeaccidents from "../components/Homeaccidents";
import Homecontact from "../components/Homecontact";
import Homenews from "../components/Homenews";
import { useState, createContext } from "react";
import { useNavigate, Link } from "react-router-dom";
import "../App.css";

export const UserLoggedInContext = createContext();

function App() {
  const [isUserLoggedIn, setUserLoggedIn] = useState(true);

  // const userNotAuthorized = (
  //   <button id="redirect-button">
  //     <Link to="/Login">Login to view Dashboard</Link>
  //   </button>
  // );

  return (
    <UserLoggedInContext.Provider value={setUserLoggedIn}>
      <BrowserRouter>
        <Routes>
          <Route path="/Login" element={<Login />} />
          <Route path="/Register" element={<Register />} />
          <Route
            path="/Homepage/about"
            element={
              isUserLoggedIn ? (
                <div>
                  <Navbar />
                  <Homeabout />
                </div>
              ) : (
                <button id="redirect-button">
                  <Link to="/Login">Login to view Dashboard</Link>
                </button>
              )
            }
          />
          <Route
            path="/Homepage/contact"
            element={
              isUserLoggedIn ? (
                <div>
                  <Navbar />
                  <Homecontact />
                </div>
              ) : (
                <button id="redirect-button">
                  <Link to="/Login">Login to view Dashboard</Link>
                </button>
              )
            }
          />
        </Routes>
      </BrowserRouter>
    </UserLoggedInContext.Provider>
  );
}
```



```

    />
    <Route
      path="/Homepage/news"
      element={
        isLoggedIn ? (
          <div>
            <Navbar />
            <Homenews />
          </div>
        ) : (
          <button id="redirect-button">
            <Link to="/Login">Login to view Dashboard</Link>
          </button>
        )
      }
    />
    <Route
      path="/Homepage/accidents"
      element={
        isLoggedIn ? (
          <div>
            <Navbar />
            <Homeaccidents />
          </div>
        ) : (
          <button id="redirect-button">
            <Link to="/Login">Login to view Dashboard</Link>
          </button>
        )
      }
    />
  </Routes>
</BrowserRouter>
</UserLoggedInContext.Provider>
);
}

export default App;

```

This above code here sets the routing for the application, manages user authentication status, and conditionally renders components based on the user authentication status. It also provides a context for managing user authentication state across the application.

Functions used:

- `import { BrowserRouter, Routes, Route } from "react-router-dom";`: Imports necessary components from `react-router-dom` library for handling routing in the application.
- Imports various components such as `Login`, `Register`, `Navbar`, `Homeabout`, `Homeaccidents`, `Homecontact`, and `Homenews`.
- `import { useState, createContext } from "react";`: Imports `useState` and `createContext` hooks from React library for managing state and creating a context.

- ``import { useNavigate, Link } from "react-router-dom";``: Imports ``useNavigate`` hook from ``react-router-dom`` for programmatic navigation and ``Link`` component for creating links within the application.
- ``import "./App.css";``: Imports a CSS file named ``App.css``, which likely contains styles for the application.
- ``export const UserLoggedInContext = createContext();``: Creates a context named ``UserLoggedInContext`` which will be used to manage the state of user authentication.
- ``function App() { ... }``: Defines the main component of the application named ``App``.
- ``const [isUserLoggedIn, setUserLoggedIn] = useState(true);``: Initializes a state variable ``isUserLoggedIn`` using the ``useState`` hook to manage the user's authentication status.
- ``<BrowserRouter>...</BrowserRouter>``: Wraps the entire application with ``BrowserRouter`` component to enable routing functionality.
- ``<Routes>...</Routes>``: Defines the routes for the application.
- ``<Route path="/Login" element={<Login />} />``: Specifies the route for the Login component.
- ``<Route path="/Register" element={<Register />} />``: Specifies the route for the Register component.
- ``<Route path="/Homepage/about" ... />``: Defines routes for different sections of the homepage (``about``, ``contact``, ``news``, ``accidents``). Based on the ``isUserLoggedIn`` state, it conditionally renders either the corresponding component or a button to redirect to the login page.

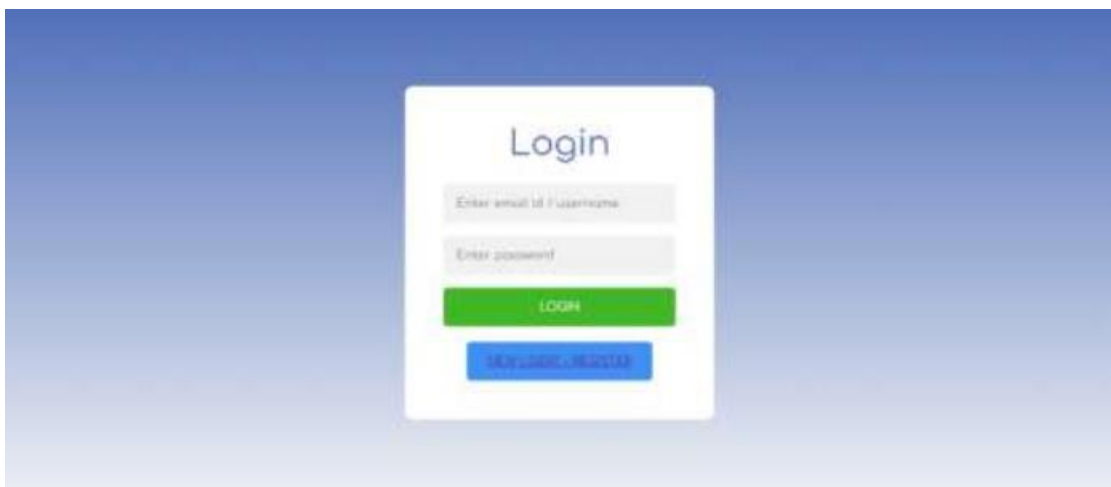


Fig.no 4.8
Login page of web

Firestore Database

The Firebase Realtime Database is a cloud-hosted database. Cloud Firestore is a flexible, scalable database for mobile, web, and server development from Firebase and Google Cloud. Like Firebase Realtime Database, it keeps your data in sync across client apps through realtime listeners and offers offline support for mobile and web so you can build responsive apps that work regardless of network latency or Internet connectivity. Cloud Firestore also offers seamless integration with other Firebase and Google Cloud products, including Cloud Functions.

```
import { initializeApp } from "firebase/app";
import { getFirestore } from "firebase/firestore";
// TODO: Replace the following with your app's Firebase project configuration
// See: https://support.google.com/firebase/answer/7015592
const firebaseConfig = {
  apiKey: "AIzaSyAlUpq-yLmFSrapiDYlxeg691CQH3NEcU",
  authDomain: "roadaccidents-harish.firebaseio.com",
  projectId: "roadaccidents-harish",
  storageBucket: "roadaccidents-harish.appspot.com",
  messagingSenderId: "641023244396",
  appId: "1:641023244396:web:e9ab2dc3ff265488808049",
};

// Initialize Firebase
const app = initializeApp(firebaseConfig);

// Initialize Cloud Firestore and get a reference to the service
const db = getFirestore(app);
export default db;
```

This above code here initializes the Firebase with the provided configuration and initializes Cloud Firestore which provides a reference to the Firestore service that can be used to interact with the Firestore database in the React application.

Functions used:

- ``import { getFirestore } from "firebase/firestore";``: Imports the ``getFirestore`` function from the Firebase Firestore module. This function is used to get a Firestore instance.
- ``const firebaseConfig = { ... };``: Defines the Firebase project configuration object. This object contains various configuration options such as the API key, authDomain, projectId, etc. These configurations are specific to your Firebase project and can be obtained from the Firebase Console.
- ``const app = initializeApp(firebaseConfig);``: Initializes Firebase with the provided configuration object (``firebaseConfig``). This function call initializes Firebase with the specified configuration options.
- ``const db = getFirestore(app);``: Initializes Cloud Firestore and obtains a reference to the Firestore service. The ``getFirestore`` function takes the initialized Firebase app (``app``) as an argument and returns a reference to the Firestore service.
- ``export default db;``: Exports the Firestore reference (``db``) as the default export. This allows other modules to import and use the Firestore reference in their code.

BACK END

Index

After done with firebase initialization we use GET endpoint to store the data in the firebase database and also deals with the request coming in. GET endpoint is an API (Application Programming Interface) endpoint which is an inbuilt javascript method which helps to interact with the backend process, as it's an inbuilt function it doesn't any installation.

```
require("dotenv").config();

const db = require("./firebase");
var cors = require("cors");
const express = require("express");
const app = express();
const PORT = process.env.SERVER_PORT || 3030;

app.use(cors());

//Another method to send data to server line 1
//const docRef = db.collection("users").doc("alovelace");

// -----get method starts here-----
app.get("/iotdblink", function (req, res) {
  console.log("get method has run");

  //Another method to send data to server line 2 -- but ID not working
  // const sendData = async (req, res) => {
  //   await docRef
  //     .set({
  //       first: "Ada",
  //       last: "Lovelace",
  //       born: 1815,
  //     })
  //     .then((docRef) => {
  //       console.log("Document written with ID: ", docRef.id);
  //     })
  //     .catch((error) => {
  //       console.error("Error adding document: ", error);
  //     });
  //   };
  //   };

  // sendData();

  db.collection("users")
    .add({
      name: req.query["name"] || "Harish",
      emergencyContact: req.query["emergencyContact"] || "+919999999999",
      address: req.query["address"] || "no.1 abc street",
      gpsLocationLat: req.query["gpsLocationLat"] || "20",
      gpsLocationLon: req.query["gpsLocationLon"] || "20",
      pressure: req.query["pressure"] || "20",
      groundClearance: req.query["groundClearance"] || "20",
    })
    .then((docRef) => {
      console.log("Document written with ID: ", docRef.id);
      res.status(200).send({
        name: req.query["name"],
        emergencyContact: req.query["emergencyContact"],
        address: req.query["address"],
        gpsLocationLatitude: req.query["gpsLocationLat"],
        gpsLocationLongitude: req.query["gpsLocationLon"],
        groundClearance: req.query["groundClearance"],
        pressure: req.query["pressure"],
      });
    });
});
```

```

        .catch((error) => {
            console.error("Error adding document: ", error);
        });

    // -----get method ends here-----
});

app.listen(PORT, () => {
    console.log(`server started on port ${PORT}`);
});

```

This above code here helps to sets up a basic Express server with a single GET endpoint that stores data in a Firestore database where it utilizes Firebase for database interaction and includes middleware to handle CORS(Cross Origin Resource Sharing).

CORS is a http header which allows server to indicate any origin other than its browser by that it permits other loading resources.

Functions used:

- ``require("dotenv").config();``: Loads environment variables from a ``.env`` file into ``process.env``, allowing sensitive data such as API keys and database credentials to be stored securely.
- ``const express = require("express");``: Imports the Express framework.
- ``const app = express();``: Creates an Express application instance.
- ``const PORT = process.env.SERVER_PORT || 3030;``: Defines the port number for the server to listen on, either from an environment variable or defaulting to port 3030.
- ``app.use(cors());``: Sets up CORS (Cross-Origin Resource Sharing) middleware to allow cross-origin requests from the client-side.
- Defines a GET endpoint ``/iotdblink`` that handles incoming requests.
- Retrieves data from the query parameters of the request and stores it in the Firestore database under the ``users`` collection.
- Responds with a status code of 200 and sends back the data received from the client as a confirmation.
- ``app.listen(PORT, () => { ... });``: Starts the Express server and listens for incoming requests on the specified port. Upon successful initialization, it logs a message indicating that the server has started and on which port it is running.

Firestore database

```
const functions = require("firebase-functions");
const admin = require("firebase-admin");
admin.initializeApp({
  credential: admin.credential.cert(require("./admin.json")),
});
const db = admin.firestore();

module.exports = db;
```

This code is setting up a Firebase Cloud Function and initializing the Firebase Admin SDK to interact with Firebase services. It initializes Firebase Cloud Functions and the Firebase Admin SDK with the necessary credentials, and it exports a reference to the Firestore database for use in other parts of the application.

The Admin SDK is a set server libraries which helps to interact with the firebase from other environment, other than java also used for many application, where as here it provides API for Real-time databases, authentication etc...

Functions used:

- `const admin = require("firebase-admin");`: Imports the Firebase Admin module, which provides administrative access to Firebase services.
- `admin.initializeApp({ ... })`: Initializes the Firebase Admin SDK with the provided configuration options.
- `credential`: Specifies the credential used to authenticate the Admin SDK. In this case, it's loading the service account credentials from a JSON file named "admin.json" located in the project directory.
- `const db = admin.firestore();`: Initializes the Cloud Firestore database using the Firebase Admin SDK. This allows the code to interact with Firestore, such as reading and writing data.
- `module.exports = db;`: Exports the Firestore database reference (`db`) to be used in other modules. This allows other modules to import and use the Firestore reference for database operations.

4.3. Constraints, Alternatives and Tradeoffs

Constraints:

Cost is one of the important constraints, as we are integrating the GPS, GSM, pressure sensor may gradually increases the upfront cost of the whole system, which can also keep increasing when are implementing it at the fleet of the vehicles.

The placement of the components which is been integrated should be at the right place and to act with in the precision of time placing it too far from the impact point leads to low accuracy and efficiency and placing it too close will leads to damage the sensors.

Sometimes it leads to over power consumption which leads to drain the power of the vehicle's battery ,so in the time implementing it needs a proper plan of balance with the functionality and the power its takes for the work which always require careful optimization.

It always require reliable data connectivity which is essential for transmitting the real time location and its data and also for sending the alert messages, sometimes in remote or low network coverage areas the connectivity issue may occur which will affect the efficiency of the system.

Alternatives:

Exploring and keep updating the technologies such as radar, lidar could provide additional or enhanced accident detection capabilities which will be used in future and also using the hybrid systems for communications like GSM with the alternate communication techs like satellite communication or mesh networks which can also improve the scope of the system even in worst mobile range.

Coming up with complete cloud based solutions for data processing and potentially reduce power consumption and the cost

Instead of separate sensors we can integrate with the existing safety measures in the vehicle systems like integrating it with the air bags deployment system or with the onboard diagnostics could improve and provide more cost effective solutions

We can propose the same system with the NB-IoT module which can be simpler hardware model because of the integrated GPS, Communication module with in the system like NRF9160 as development kit with Nordic thingy:91, this has different keywords compared to embedded C and the communication will get connected all over the globe.

Trade-offs:

Increasing the number of sensors and components for the greater good can be done but it can also leads to the other way like malfunction of the sensors and over power consumption and few more constraints as mentioned before.

Investing in high quality sensor and other modules improves the stability and efficiency of the system and leads to increase in the overall costs for the implementation and the other stuffs.

Processing of the data like transmitting the real time location coordinates which leads to some privacy concerns of the user, so the balancing of the essential need of accurate accident detection with the user privacy rights are also essential.

Have to follow government norms is always essential because in the proposal system like this there may arise compliance with regulatory requirements, such as data protection law and automotive safety standards, this may push the constraints on the system design and the operation which leads to a alternate solution.

5. SCHEDULE, TASKS AND MILESTONES

Table 5.1
TIMELINE

SCHEDULE	TASKS	MILESTONES
January	Finalizing the topic and understanding it and starts to work on it and trying to figure out the prototype.	Zeroth review
February	Deciding the components and stimulation of circuit with the integration components and initializing the code for each components and developing the prototype.	Review 1
March	Beginning stage of integrating the sensors and the modules with to work as complete prototype and as proposed system and cross checking with the each components performances and also working additional web application	Review 2
April	To add additional feature working on full stack web application(in-built) and getting the codes for the software and get integrated with the prototype and preparation of report	Report submission
May	As per guidelines preparing presentation for final review	Review 3

6. RESULTS AND DISCUSSION

6.1.HARDWARE OUTPUTS:

Ultrasonic sensor:

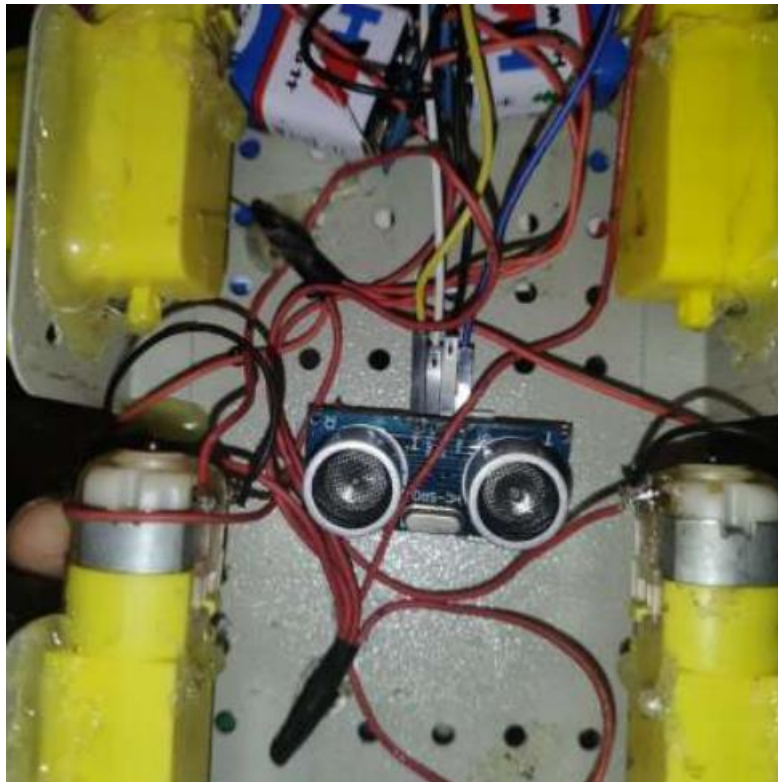
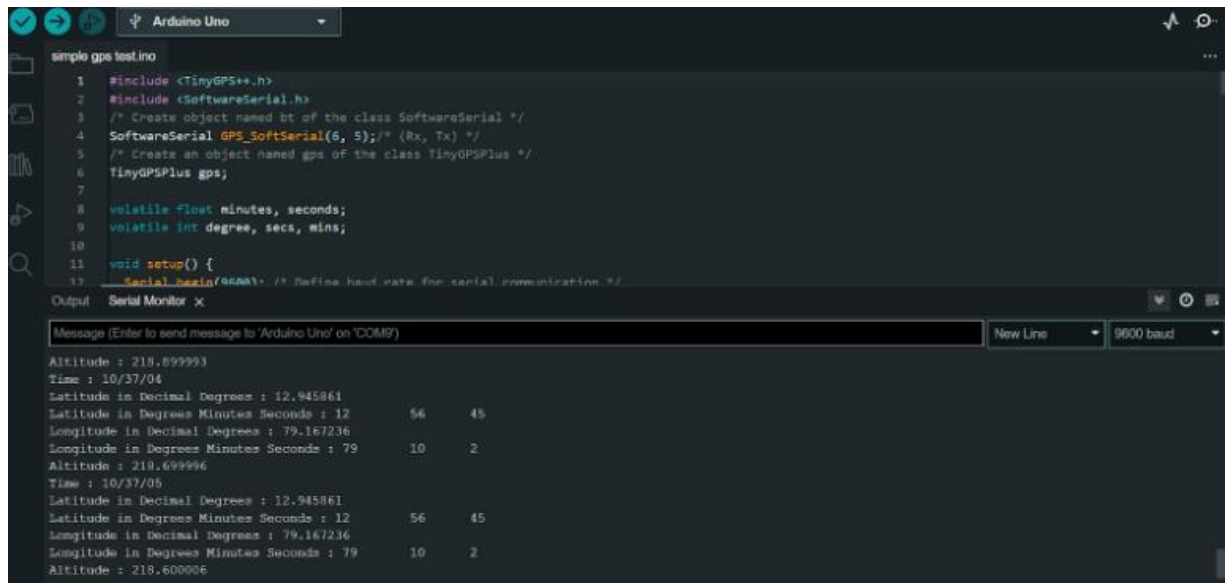


Fig.no 6.1
Ultrasonic sensor implementation

As said earlier here it is used for ground clearance



```
1 #include <TinyGPS++.h>
2 #include <SoftwareSerial.h>
3 /* Create object named bt of the class SoftwareSerial */
4 SoftwareSerial GPS_SoftSerial(6, 5); /* (Rx, Tx) */
5 /* Create an object named gps of the class TinyGPSPlus */
6 TinyGPSPlus gps;
7
8 volatile float minutes, seconds;
9 volatile int degree, secs, mins;
10
11 void setup() {
12   Serial.begin(9600); /* Define baud rate for serial communication */
13 }
```

Output Serial Monitor x

Message (Enter to send message to 'Arduino Uno' on 'COM9') New Line 9600 baud

Altitude : 218.699993
Time : 10/37/04
Latitude in Decimal Degrees : 12.945861
Latitude in Degrees Minutes Seconds : 12 56 45
Longitude in Decimal Degrees : 79.167236
Longitude in Degrees Minutes Seconds : 79 10 2
Altitude : 218.699996
Time : 10/37/05
Latitude in Decimal Degrees : 12.945861
Latitude in Degrees Minutes Seconds : 12 56 45
Longitude in Decimal Degrees : 79.167236
Longitude in Degrees Minutes Seconds : 79 10 2
Altitude : 218.600006

Fig.no 6.4
GPS output

We can see the output coordinates which has been taken by the GPS

Output:

Latitude in Decimal Degrees : 12.945850

Latitude in Degrees Minutes Seconds : 12 56 45

Longitude in Decimal Degrees : 79.167221

Longitude in Degrees Minutes Seconds : 79 10 1

Altitude : 219.600006

Time : 10/38/39

Pressure sensor:



Fig.no 6.5
Pressure sensor implementation

After the ground clearance of the pressure sensor we get the output



Fig.no 6.6
Pressure sensor output

GSM module:



Fig.no.6.7
GSM module implementation

Connection of GSM module to the network to send http request

6.2.SOFTWARE OUTPUTS:

Web application:

Frontend contact page

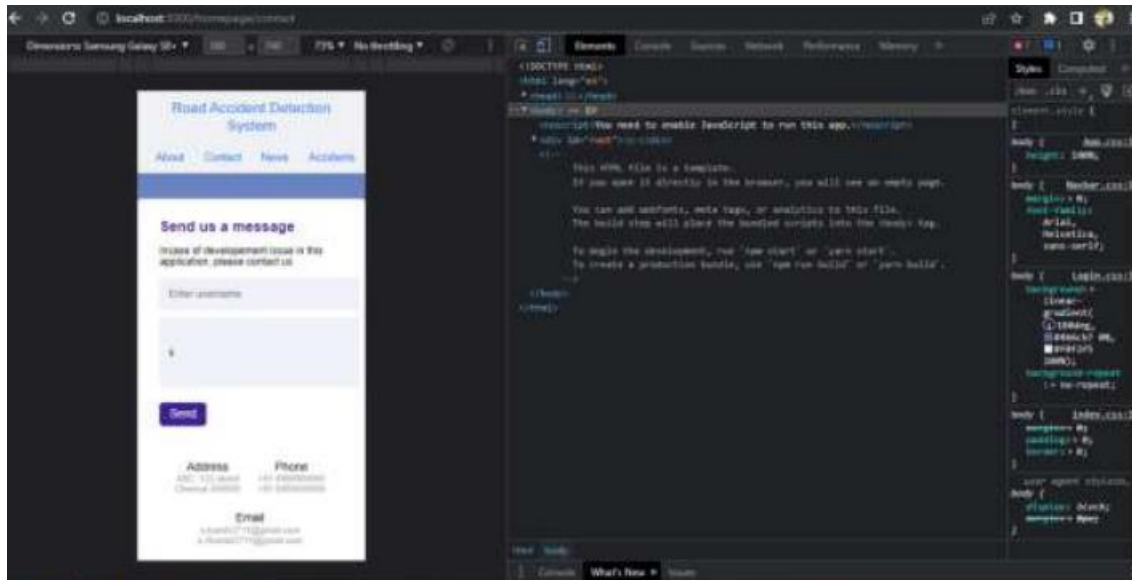


Fig.no 6.8
Frontend responsive page

Database – Firebase

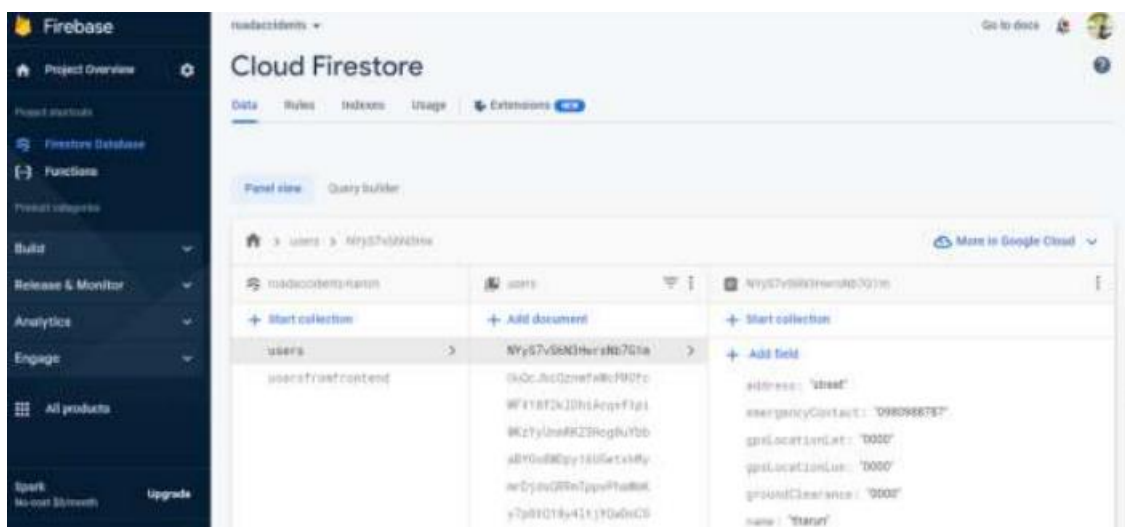


Fig.no 6.9
Cloud firestore

Redirector

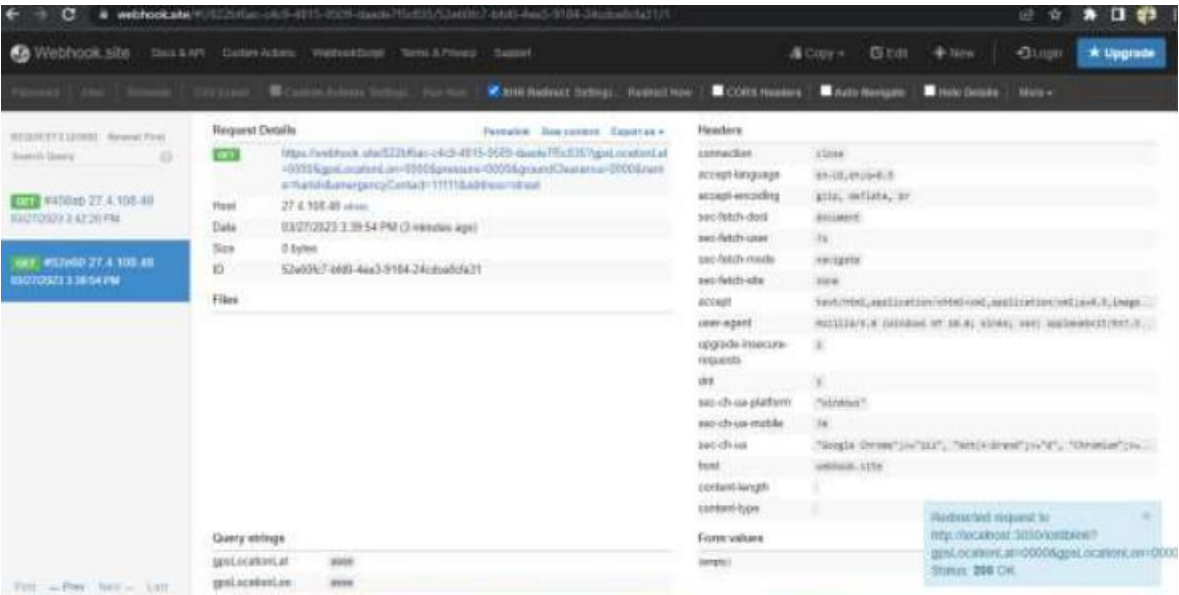


Fig.no 6.10
Intermediate redirector

FINAL OUTPUT

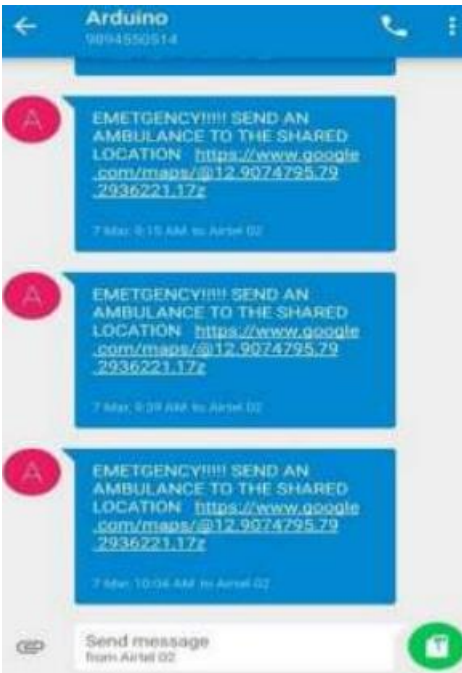


Fig.no 6.11
Mobile SMS

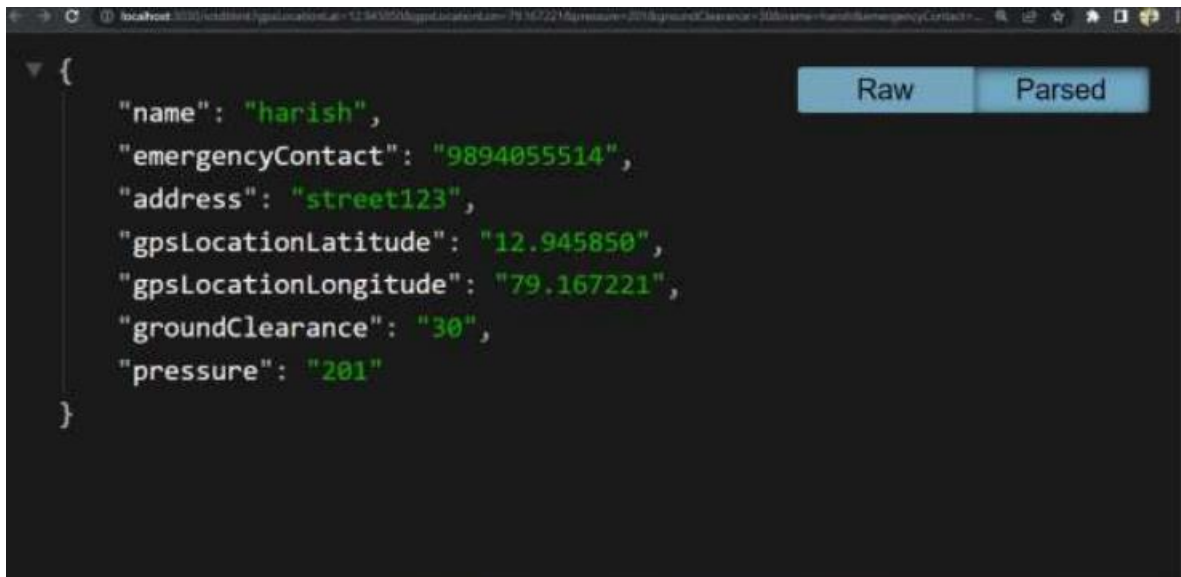


Fig.no 6.12
API endpoint

6.3.PROJECT DEMONSTRATION AND DISCUSSION :

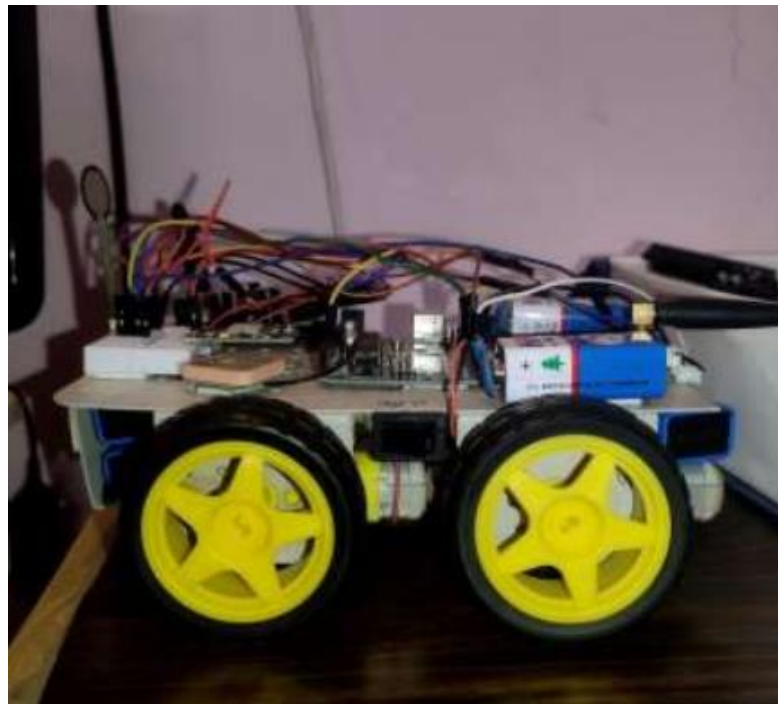


Fig.no 6.13
Prototype of car

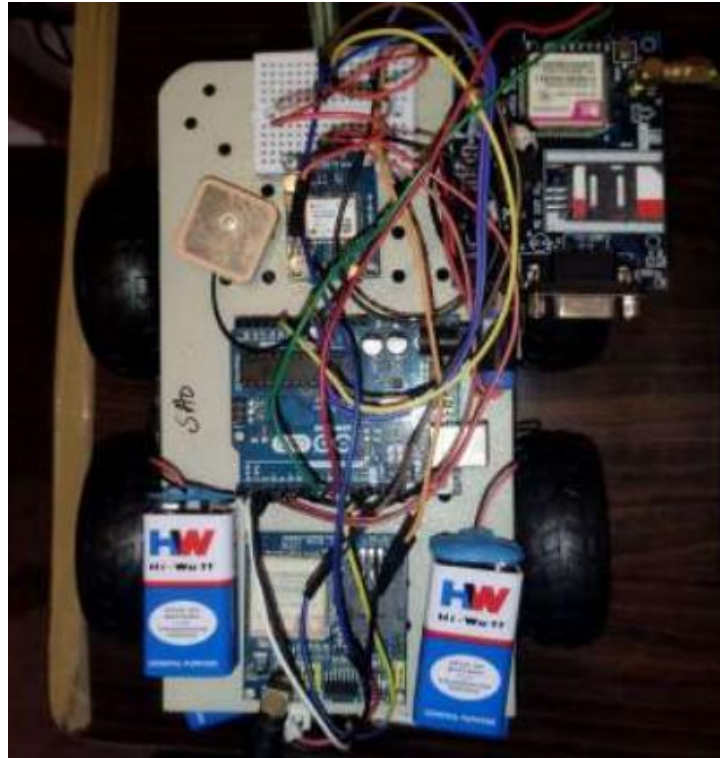


Fig.no 6.14
Installation of components based on circuit

Video of the working prototype:

<https://drive.google.com/file/d/1D4icxi73kB6yCcNHd6wNObIw8bSdAm8n/view?usp=drivesdk>

As we said earlier the proposed system three modules which consists of the above mentioned sensor and the web application gives output with the good accuracy we can get the accuracy of the GPS than the existing system which has been used with MEMS and also have low accuracy in less traffic areas we can detect it mostly all places with the arduino so we get better output accuracy than the existing system.

7. SOCIAL IMPACT

By improving the accuracy and the detection and prevention of the accident it helps in reduce the number of accidents and the casualties and the damage that will be caused due to it. Faster detection and alert messages will leads to the faster response from the medical assistance which minimize the injuries

Prevention of the false alerts as we overcome gyroscope sensors because it can call a false alert when you are travelling in slope due to the increase in the altitude, so we overcome with ultrasonic sensor.

The deployment of a advanced accident and prevention technologies can increase public confidence in vehicle safety system. This may inspire more details to adopt and can utilize the system which leads increasing the societal benefits in over all road safety prespective

The data which has been collected like the frequency of the accident, severity and location which can provide valuable things for the people like urban planners and also these information helps to identify the accident prone areas and can implement some targeted interventions like improvement on traffic management which helps to enhance the road safety.

8. SUMMARY

The project proposal aims to create a system leveraging IoT technology to mitigate fatalities resulting from accidents. It utilizes an array of sensors including ultrasonic, pressure, GPS, and GSM to detect accidents and promptly notify emergency services about the accident's location and severity. This initiative underscores IoT's capacity to address real-world challenges and enhance safety standards.

The system's integration with vehicles via OEM streamlines its deployment and usability, potentially leading to life-saving outcomes through timely communication with emergency services and minimizing accident impacts. In essence, this project carries substantial implications for public safety, demonstrating IoT's transformative potential in bolstering safety measures.

The creation of an IoT system utilizing Arduino UNO for ensuring safe employee commuting stands as a significant project. Its relevance to the organization is paramount as it prioritizes the safety of employees during their travel to and from the office using personal vehicles. The system's capability to swiftly detect accidents and notify emergency services via GPS and GSM technology holds the promise of potentially saving numerous lives.

By dispatching an alert to the emergency hotline and monitoring a corresponding web application, the system facilitates prompt dispatch of ambulance services to accident sites. This rapid response can be pivotal in preserving lives, as immediate medical attention significantly enhances survival prospects for accident victims. The transmitted alert includes real-time accident location and impact severity, aiding emergency services in assessing the urgency of the situation.

Integration of pressure sensors enables the system to gauge accident severity, assisting in administering appropriate medical aid to affected individuals. Timely medical intervention is crucial, as delays can exacerbate injuries and escalate fatality risks. Hence, the system's ability to promptly detect accident severity and notify emergency services stands as a vital asset in life-saving endeavors.

Overall, the system holds potential to enhance employee safety and mitigate accident repercussions, exemplifying the transformative capacity of IoT technology in addressing tangible real-world challenges.

9. REFERENCES

- [1] Vehicle Accident Detection & Alert System using IoT and Artificial Intelligence-
<https://ieeexplore.ieee.org/document/9544940>
- [2] IOT based detection of vehicle accident and an emergency help using application interface- <https://link.springer.com/article/10.1007/s00542-023-05414-z>
- [3] A comprehensive study on IOT based accident detection systems for smart vehicles-<https://ieeexplore.ieee.org/iel7/6287639/6514899/09133106.pdf>
- [4] IoT based car accident detection and notification algorithm for general road accidents-<https://core.ac.uk/download/pdf/329119566.pdf>
- [5] Car Accident Detection and Car Health Monitoring using IoT-
https://www.ijresm.com/Vol.2_2019/Vol2_Iss4_April19/IJRESM_V2_I4_6.pdf
- [6] A Novel Internet of Things-Enabled Accident Detection and Reporting System for Smart City Environments - <https://www.mdpi.com/1424-8220/19/9/2071>
- [7] An IOT Based Smart System for Accident Prevention and Detection -
<https://ieeexplore.ieee.org/abstract/document/8697663>
- [8] Preventing Drunken Driving Accidents using IoT-
<https://openurl.ebsco.com/EPDB%3Aagcd%3A12%3A10112204/detailv2?sid=ebsco%3Aplink%3Ascholar&id=ebsco%3Aagcd%3A122961255&crl=c>
- [9] An Internet of Things(IoT) based smart helmet for accident detection and notification - <https://ieeexplore.ieee.org/abstract/document/7839052>
- [10] Accident detection and reporting system using GPS, GPRS and GSM technology-
<https://ieeexplore.ieee.org/abstract/document/6317382>
- [11] Intelligent Accident-Detection And Ambulance- Rescue System -
<https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=a1be42c6503428989ebb708db0dd39323c3cf02b>
- [12] Automatic accident detection and ambulance rescue with intelligent traffic light system -
<https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=1cd80c89c3529231d8196d73387563af047f3085>
- [13] Smart vehicle accident detection and alarming system using a smartphone -
<https://ieeexplore.ieee.org/abstract/document/7399319>