A project report for partial fulfilment of the degree of Bachelor of Technology in Electronics and Communication Engineering on

"AUTOMATIC CRASH DETECTION AND EMERGENCY SOS"

Submitted by

NAME	ROLL NUMBER	YEAR	DEPARTMENT
ANSHUK TALUKDAR	18701620033	4TH YEAR	ELECTRONICS AND COMMUNICATION ENGINEERING
MD MODASSIR ALAM	18701620034	4TH YEAR	ELECTRONICS AND COMMUNICATION ENGINEERING
LOKADITYA GHOSH	18700319048	4TH YEAR	ELECTRONICS AND COMMUNICATION ENGINEERING
SNEHASISH MUKHERJEE	18700319016	4TH YEAR	ELECTRONICS AND COMMUNICATION ENGINEERING

Under supervision the of

Prof. ARUP KUMAR GHOSH
Assistant Professor

Department of Electronics and Communication Engineering

Techno International New Town (formaly known as Techno India College of Technology) 1/1, Service Rd, DG Block(Newtown), Action Area I, Newtown, Chakpachuria, New Town, West Bengal 700156

TECHNO INTERNATIONAL NEW TOWN

(Formerly known as Techno India College of Technology)

Block - DG 1/1, Action Area 1, New Town, Kolkata - 700156, West Bengal, India
Phone: +91-33-2324-2050, 2324-2090, 2324-2091

CERTIFICATE

This is to certify that **MD MODASSIR ALAM, ANSHUK TALUKDAR, LOKADITYA GHOSH, SNEHASHISH MUKHERJEE** of the department of Electronics and Communication Engineering have successfully completed a project on **AUTOMATIC CRASH DETECTION AND EMERGENCY SOS** during their 4th year in BACHELORS OF TECHNOLOGY, session 2020-2024 at TECHNO INTERNATIONAL NEW TOWN under MAULANA ABUL KALAM AZAD UNIVERSITY OF TECHNOLOGY

03.06.2024

Head of the Dept.

Prof (Dr.) Pradip Kumar Ghosh
Department of Electronics and Communication Engineering

Techno International New Town
(Formerly Techno India College of Technology)
Block DG, Action Area-1/1,
New Town, Kolkata-700 156

3.6.24

Supervisor

(Prof. Arup Kumar Ghosh)

Acknowledgement

It is our profound privilege to express our deep sense of gratitude to Techno International New Town. We would like to thank **Prof (Dr.) Pradip Kumar Ghosh**, Head of the Department of Electronics and Communication Engineering for providing all possible support without which this project could not be accomplished. We would also like to thank our mentor, **Prof Arup Kumar Ghosh** for his guidance and encouragement that helped us a lot to pursue the project. Without his/her support, this attempt would never have been successfully implemented, we owe him a debt of gratitude.

Our parents have always been supportive. Their blessings are the roots in our endeavors. Finally, we would like to acknowledge our departmental faculty members for being the source of inspiration and kind cooperation to the completion of our work.

Signature

MD MODASSIR ALAM (18701620034)

LOKADITYA GHOSH (18700319048)

SNEHASISH MUKHERJEE (18700319016)

ANSHUK TALUKDAR (18701620033)

Department of Electronics and Communication

Techno International New Town

CONTENTS

AUTOMATIC CAR CRASH DETECTION AND SOS

Abstract

This project highlights the critical role of automation in SOS systems during car crashes, emphasizing its significance for rapid emergency response. Envisioning a scenario where automation seamlessly connects with emergency services through a System of Systems (SOS), the benefits become evident. The key advantages include swift notifications, precise location identification using advanced GPS technologies, real-time communication for immediate aid, and automated data analysis for superior evaluation.

Furthermore, the model underscores the integration of vehicle systems for automatic emergency calls, reducing reliance on manual intervention. The exchange of real-time data among different components of the emergency response system leads to a more efficient and synchronized response. Security and privacy considerations are acknowledged, recognizing the need for safeguarding confidential data collected during such events.

Lastly, the cohesion of the current infrastructure is highlighted, emphasizing the integration of existing emergency services infrastructure to ensure consistent and standardized protocols and technologies. This abstract encapsulates the multifaceted benefits and considerations associated with the integration of automation in SOS systems, showcasing its potential to significantly enhance the efficiency and effectiveness of emergency response efforts during car crashes.

Objective of the proposed system

The primary objective of this project is to design, implement, and deploy a cost-effective and universally applicable Automated SOS System for vehicles. The system aims to democratize advanced safety features by utilizing an affordable sensor for real-time crash data collection. This data will be processed by a microcontroller, which will autonomously determine the severity of the impact.

In instances of non-fatal crashes, the microcontroller will issue immediate warnings to drivers, enhancing overall road safety awareness. For fatal crashes, the system will trigger an SOS signal through a dedicated mobile application. This mobile application, designed to be easily integrated into both new and second-hand vehicles, will leverage pre-inputted data and location services to promptly dispatch SOS alerts to the nearest relief services.

The project's key focus is on breaking the existing trend of Automated SOS systems being exclusive to high-end vehicles. The objective is to create a system that is not only cost-effective but also easily integratable into a wide range of vehicles, including those prevalent in regions like India, where second-hand cars are commonly used. By achieving this objective, the project aims to contribute significantly to enhanced road safety on a broader scale.

Literature Survey

Authors	Topic	Key points	Limitations
Daghan Dogan1 , Seta Bogosyan2 and Tankut Acarman3	Evaluation of driver stress level with survey, galvanic skin response sensor data, and force-sensing resistor data	FSR sensor data were collected from the prototype electric car to assess the stress levels of drivers. FSR sensors measure force or pressure applied to them, providing insights into the stress experienced by drivers during the specified route.	FSR sensors may be sensitive to external factors such as road conditions, vehicle vibrations, and other environmental variables, potentially influencing stress level measurements.
Tommaso Addabbo ,Ada Fort ,Marco Mugnaini	Using the I2C bus to set up Long Range Wired Sensor and Actuator Networks in Smart Buildings	I2C is known for its simplicity, making it easy to implement and suitable for applications where a straightforward communication protocol is sufficient. I2C typically requires only two wires (SDA and SCL), which helps conserve GPIO pins on microcontrollers and reduces the physical footprint of the interconnection.	I2C is typically designed for short-distance communication within a circuit or on a PCB. While the described architecture claims a transmission range of up to 100 meters, this is not common for standard I2C implementations. Longer distances may require additional considerations or the use of other communication protocols
Nazir Ahmmed ,Nusrat Jahan Jenny ,Most Fowziya Akther	VADet: An Arduino based automated vehicle accident detection	Enhancement of the notification system to ensure more accurate and timely alerts. This includes	One of the challenges is the potential for false positives, where the system may incorrectly interpret certain events or

Houya ,Anika Ibnat Binte Alam , Md. Adnan Arefeen	and messaging system	sending messages to the closest response sites such as nearest police stations and hospitals	movements as accidents. This can lead to unnecessary emergency responses and may strain emergency services. Conversely, there is a risk of false negatives, where the system fails to detect an actual accident. This could occur in situations where the impact is subtle or the triggering conditions are not met, leading to delayed or no response in critical situations.
A. S. Sadun, J. Jalani, J.A. Sukor	Force Sensing Resistor (FSR): A Brief Overview and the Low Cost Sensor for Active Compliance Control	The paper underscores the significance of a well-designed sensor cover, particularly when dealing with objects of different shapes. A proper sensor cover design is crucial for optimizing contact force measurement accuracy, highlighting the need for careful consideration of the physical interaction between the sensor and the objects being measured.	FSR sensors may exhibit sensitivity variations when interacting with uneven surfaces or objects with irregular shapes. The reliability of force measurements can be compromised in such scenarios, highlighting a limitation in their applicability.
Shaikh Khalid, Mudassir Khan, Soumya Sharma, Suyog Parkhe	Design and Development of Sensor in Measuring Force	Achieve accurate force measurements is crucial for understanding the impact on comfort. Caliberating and validating force sensor against a standardized referenceis essential for reliability.	Force sensors themselves may have inherent variability, and their accuracy can be affected by factors such as temperature, humidity, and aging. Calibration against a standardized reference helps, but variations in sensor performance may still exist.

Proposed Model

Reference Diagram –

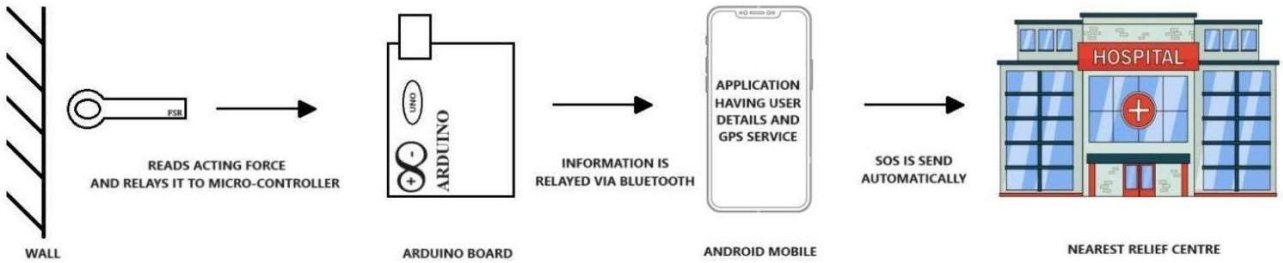


Fig 1

Methodology –

Our whole model can be understood through the below steps –

- The force sensor embedded in the car's structure detects the force generated by a collision.
- The detected force information is transmitted to the Arduino Uno board, capable of receiving and processing this data.
- The Arduino Uno, pre-programmed with algorithms, evaluates the received force data to determine the severity of the impact.
- For a minor impact, a blinking LED informs the driver. In the case of a severe crash, the system triggers a Bluetooth hub.
- The Bluetooth hub establishes a wireless connection with the driver's Android device.
- Arduino code sends specific inputs to the Android device through the Bluetooth module instead of displaying results on a screen.
- The driver's mobile application, upon registration, captures vital information about the driver, including vehicle registration number, name, blood group, etc.
- In the event of a severe crash, the mobile application compiles all driver information into an SOS alert.
- A fixed interval allows the driver to use a kill switch, terminating the SOS process if relief services are not required.
- If the SOS process is not terminated within the specified interval, the application automatically determines the driver's location using the phone's location services. It then identifies and locates the nearest hospital and other relief-providing institutions.
- This integrated workflow ensures a comprehensive impact detection system that communicates impact severity visually to the driver and seamlessly integrates Bluetooth connectivity with the Android device for advanced communication and automatic SOS functionalities.

Hardware Components -

- Arduino UNO
- Force Sensitive Resistor | FSR- 408, 50mm diameter, 18 Kohm
- 2x16 display unit
- I2C module
- Jumper wires
- LED
- 1 Kohm resistor
- Breadboard.
- 9-volt battery (power source)
- Bluetooth Hub

Additionally, an RC car is also constructed as a part of demonstration, on which the whole setup is mounted. The RC car takes in power input from the AC socket which by using an eliminator is rectified and stepped down to run a 12v dc motor for the wheels.

Some of the important components are listed below:-

Arduino UNO	Arduino is an open-source electronics platform It utilizes a programmable microcontroller for project development. Arduino provides a user-friendly integrated development environment (IDE) Allows programming in the Arduino programming language.
I2C module for LCD	Utilizes I2C (Inter-Integrated Circuit) communication protocol. Enables simplified wiring by reducing the number of connections. Employs two lines, SDA (data) and SCL (clock), for efficient data transfer, which for Arduino uno is fixed as A4 and A5 analog ports respectively Requires fewer pins on the microcontroller, optimizing available pins for other components

FSR	FSR is a sensor that detects pressure or force applied to its surface. Exhibits variable resistance based on the magnitude of the applied force. Provides an analog output corresponding to the force applied. Capable of precise force measurement in various applications. Responds dynamically to changes in applied force, offering real-time feedback.

Project Progress

Reference Diagram –

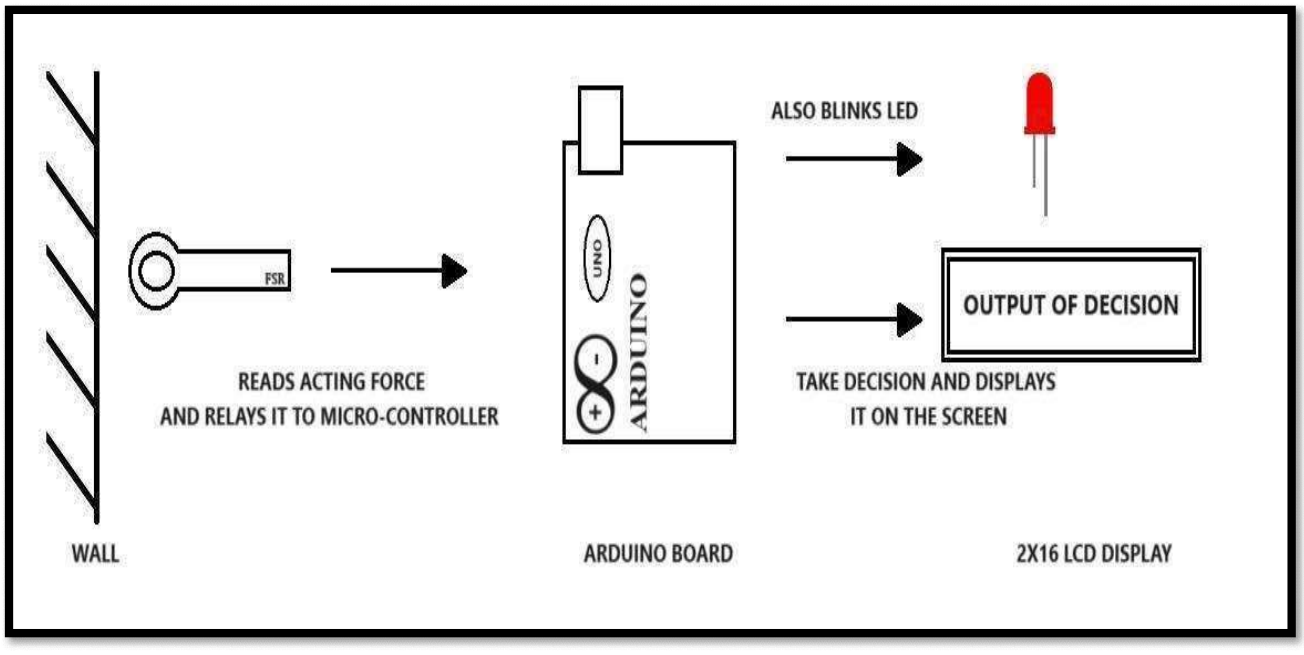


Fig 2

Arduino UNO: -

The Arduino UNO is a popular microcontroller board based on the ATmega328P. It is widely used in electronics projects and prototyping due to its ease of use and versatility. Below is a detailed description of its features, power sources, and pin configurations.

Fig 3

Key Features
Microcontroller: ATmega328P
Operating Voltage: 5V
Input Voltage (recommended): 7-12V
Input Voltage (limits): 6-20V
Digital I/O Pins: 14 (of which 6 provide PWM output)
Analog Input Pins: 6
DC Current per I/O Pin: 20 mA
DC Current for 3.3V Pin: 50 mA
Flash Memory: 32 KB (ATmega328P) of which 0.5 KB used by bootloader
SRAM: 2 KB (ATmega328P)
EEPROM: 1 KB (ATmega328P)
Clock Speed: 16 MHz
LED_BUILTIN: Pin 13
Dimensions: 68.6 mm x 53.4 mm
Weight: 25 g
Power Sources
The Arduino UNO can be powered via:

USB Connection: Provides 5V and is commonly used during development and prototyping.
Barrel Jack: Accepts 7-12V input from an AC-to-DC adapter or battery.
Vin Pin: Can be used to supply the board with an external voltage source, typically between 7-12V.
The board also features an onboard voltage regulator to ensure stable 5V and 3.3V outputs.

Pin Configuration
The Arduino UNO includes a variety of pins for different functionalities:

Digital Pins (0-13):

Can be used as input or output.
Pins 3, 5, 6, 9, 10, and 11 support PWM output.
Analog Pins (A0-A5):

Used to read analog signals.
Can also be used as digital pins.
Power Pins:

Vin: Input voltage to the Arduino board when using an external power source.
5V: Provides a regulated 5V output.
3.3V: Provides a regulated 3.3V output.
GND: Ground pins.

Reset Pin: Can be used to reset the microcontroller.

Serial Communication Pins:

TX (1): Transmit data.
RX (0): Receive data.
SPI Pins:

MOSI (11): Master Out Slave In.
MISO (12): Master In Slave Out.
SCK (13): Serial Clock.
I2C Pins:

SDA (A4): Serial Data Line.
SCL (A5): Serial Clock Line.
Circuit Diagram
The Arduino UNO's circuit integrates all these features, with connections to the microcontroller's pins clearly labeled. Key components include the voltage regulators, USB interface, and the ATmega328P microcontroller itself.

The onboard LED connected to pin 13 is a simple way to test the board's functionality, often used in the "blink" sketch.

Uses of Arduino UNO
The Arduino UNO is a versatile microcontroller board that serves a wide range of applications. Its ease of use, extensive support, and community resources make it ideal for both beginners and experienced developers. Here are some common uses:

1. Educational Purposes
Learning Programming and Electronics: Arduino UNO is widely used in educational settings to teach students the basics of programming, electronics, and circuit design.
STEM Projects: It supports STEM (Science, Technology, Engineering, and Mathematics) education through hands-on projects and experiments.
2. Prototyping and Development
Rapid Prototyping: Engineers and hobbyists use the Arduino UNO to quickly prototype new ideas and projects.
Proof of Concept: It is used to demonstrate the feasibility of a concept before committing to full-scale production.
3. Home Automation
Smart Home Projects: The Arduino UNO can be used to control lights, thermostats, security systems, and other devices in a smart home setup.
IoT Integration: It can be integrated with various IoT (Internet of Things) platforms to create connected devices.
4. Robotics
Robot Control: The board is commonly used to control robots, including mobile robots, robotic arms, and drones.

Sensors and Actuators: It interfaces with various sensors (e.g., temperature, humidity, light) and actuators (e.g., motors, servos) to create responsive robotic systems.

5. Wearable Technology
Wearable Devices: Arduino UNO can be used in the development of wearable technology, such as fitness trackers and smart clothing.
Portable Projects: Its small size and lightweight nature make it suitable for portable electronic projects.
6. Art and Interactive Installations
Interactive Art: Artists use Arduino UNO to create interactive installations that respond to environmental stimuli.
Sound and Light Shows: It can control LEDs, speakers, and other output devices for dynamic visual and auditory displays.
7. Environmental Monitoring
Weather Stations: It can be used to build weather stations that monitor temperature, humidity, and atmospheric pressure.
Air Quality Monitoring: Arduino UNO can interface with air quality sensors to track pollution levels and other environmental factors.
8. Automated Systems
Industrial Automation: The board can be used in industrial applications to automate machinery and processes.
Agricultural Automation: It is used in smart farming projects to automate irrigation, monitor soil moisture, and control greenhouse environments.
9. Data Logging
Sensor Data Collection: Arduino UNO can log data from various sensors for analysis and research purposes.
Real-Time Monitoring: It enables real-time monitoring of systems and environments, which is valuable in scientific experiments and environmental studies.
10. DIY Projects

Custom Gadgets: Hobbyists use Arduino UNO to create custom gadgets and inventions tailored to their specific needs and interests.

Maker Community Projects: It is a central component in many DIY projects shared within the maker community, fostering innovation and collaboration.

Force Sensitive Register (FSR): -

Force Sensitive Resistor (FSR): Features and Specifications
A Force Sensitive Resistor (FSR) is a type of sensor that changes its resistance based on the force or pressure applied to its surface. These sensors are commonly used in applications that require force or pressure measurement.

Fig 4

Key Features
Pressure Sensitivity:

Resistive Range: The resistance decreases as the force on the FSR increases.
Force Range: Typically, FSRs can sense forces ranging from a few grams to several kilograms.
Size and Shape:

Available in various sizes and shapes, commonly circular or square.
Thin and flexible, making them easy to integrate into various surfaces.
Response Time:

Fast response time, usually less than 10 milliseconds.
Suitable for real-time force or pressure monitoring.
Durability:

Designed to withstand multiple actuations.
Can be used in rugged environments.
Temperature Range:

Operates effectively within a temperature range, typically from -30°C to +70°C.
Analog Output:

Provides an analog output that corresponds to the amount of force applied.
Can be interfaced with analog-to-digital converters for digital signal processing.
Power Source
FSRs do not have an intrinsic power source. They are passive components and require an external voltage source when used in a circuit. Typically, they are connected in a voltage divider configuration where the FSR and a fixed resistor create a voltage output that varies with the applied force.

Pin Configuration
FSRs usually have two terminals:

Terminal 1: Connects to one side of the resistive element.
Terminal 2: Connects to the other side of the resistive element.
Circuit Configuration
To use an FSR, it is typically set up in a simple voltage divider circuit. Here is a basic example:

Voltage Divider Circuit:
Components: FSR, fixed resistor, power supply (usually 5V), and a microcontroller or ADC (Analog-to-Digital l Converter) pin.

Configuration:
Connect one terminal of the FSR to the power supply (5V).
Connect the other terminal of the FSR to one terminal of the fixed resistor.
Connect the other terminal of the fixed resistor to ground (GND).
The junction between the FSR and the fixed resistor provides the output voltage, which varies based on the applied force.
This output voltage is read by an analog input pin of a microcontroller (e.g., Arduino) or an ADC.
Operation:

When no force is applied, the FSR has high resistance, resulting in a low voltage at the junction.
As force increases, the resistance of the FSR decreases, causing the voltage at the junction to increase.
This varying voltage can be measured and interpreted by the microcontroller or ADC to determine the amount of force applied.
Example Circuit Diagram
css
Copy code
Vcc (5V)
|
|
[FSR]
|
|----- Analog Input (to microcontroller)
|
[Fixed Resistor]
|
|
GND
In this setup, the microcontroller reads the analog input voltage, which varies based on the force applied to the FSR. This voltage can then be converted to a force measurement using calibration data specific to the FSR being used.

TOUCH SENSOR

A touch sensor is a type of sensor that detects the presence and location of a touch within a given area. These sensors are widely used in consumer electronics, industrial controls, and interactive devices.

Key Features
Touch Detection:

Capacitive or Resistive: Most touch sensors operate based on capacitive or resistive technology.
Multi-touch Capability: Some sensors can detect multiple touch points simultaneously.
Sensitivity:

High Sensitivity: Capable of detecting light touches and variations in touch pressure.
Adjustable Thresholds: Sensitivity levels can often be adjusted to suit different applications.
Response Time:

Fast Response: Typically respond to touch within a few milliseconds, providing real-time interaction.
Durability and Robustness:

Long Lifespan: Designed for numerous touch cycles.
Rugged Design: Often built to withstand harsh environments and frequent use.
Interface Options:

Digital Output: Most touch sensors provide a digital output (HIGH/LOW) indicating touch detection.
Analog Output: Some advanced touch sensors provide an analog output corresponding to the touch pressure or position.
Low Power Consumption:

Energy Efficient: Suitable for battery-powered applications due to low power requirements.
Power Source
Touch sensors require a power source to operate. Typically, they operate at a voltage range of 3.3V to 5V, making them compatible with most microcontroller platforms such as Arduino.

Pin Configuration
The pin configuration for a basic touch sensor usually includes:

VCC: Connects to the power supply (3.3V or 5V).
GND: Connects to ground.
OUT: Provides the digital output signal (HIGH when touched, LOW when not touched).
For sensors with additional features, there may be more pins, such as an analog output or communication pins (I2C, SPI) for more complex touch sensors.

Circuit Configuration
A typical touch sensor circuit is straightforward. Here's how to connect a basic touch sensor to a microcontroller:

Power Supply Connection:

VCC: Connect to the 3.3V or 5V pin on the microcontroller.
GND: Connect to the ground (GND) pin on the microcontroller.
Output Signal:

OUT: Connect to a digital input pin on the microcontroller to read the touch status.
Example Circuit Diagram
sql
Copy code

```
 Microcontroller
   +----------------+
   |                |
   |  [Touch Sensor]  |
   |  VCC --- 3.3V/5V |
   |  GND --- GND   |
```

| OUT --- Digital Pin (e.g., D2) |
+----------------+

Operation

Power Up:

When powered, the touch sensor continuously monitors its surface for touch inputs.

Touch Detection:

When a touch is detected, the sensor's OUT pin goes HIGH.

The microcontroller reads this HIGH signal as an indication of touch.

Touch Release:

When the touch is released, the OUT pin goes LOW.

The microcontroller reads this LOW signal indicating no touch.

Applications

Consumer Electronics:

Touchscreens: Used in smartphones, tablets, and laptops.

Home Appliances: Touch panels on ovens, refrigerators, and other appliances.

Industrial Controls:

Control Panels: Used in machinery for user input and control.

Safety Systems: Touch-sensitive safety features in equipment.

Interactive Devices:

Kiosks: Self-service terminals and ATMs.

Wearable Devices: Smartwatches and fitness trackers.

Automotive:

Infotainment Systems: Touch controls for navigation and media.

Dashboard Controls: Touch-sensitive buttons and switches.

BLUETOOTH MODULE

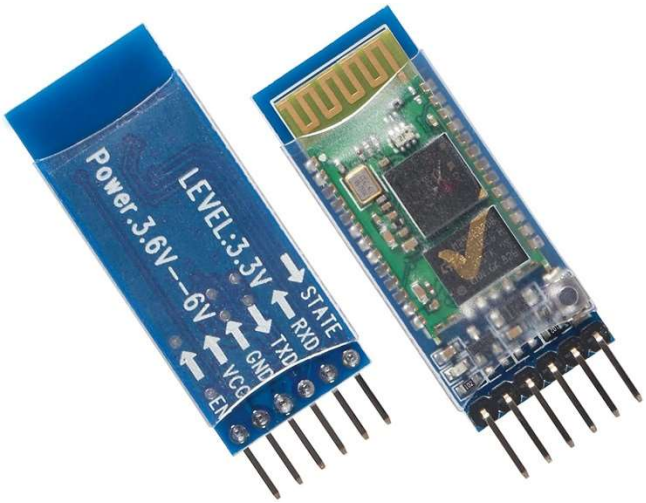


Fig 5

A Bluetooth module is a device that enables wireless communication between electronic devices over short distances using Bluetooth technology. These modules are commonly used in various applications, from simple data transfer to complex wireless control systems.

Key Features
Wireless Communication:

Bluetooth Standard: Typically supports Bluetooth 2.0, 3.0, 4.0 (BLE), or newer versions.
Range: Usually operates within a range of 10 meters (Class 2) or up to 100 meters (Class 1), depending on the module.
Data Rate:

Transfer Speed: Varies by module and Bluetooth version, commonly around 1-3 Mbps for classic Bluetooth and up to 2 Mbps for Bluetooth Low Energy (BLE).
Compatibility:

Multi-platform Support: Compatible with various operating systems like Windows, Android, iOS, and Linux.
Serial Communication: Interfaces easily with microcontrollers and PCs via UART (Universal Asynchronous Receiver-Transmitter).
Power Consumption:

Low Power Mode: Many modules support low power modes, making them suitable for battery-operated devices.
Operating Voltage: Typically operates at 3.3V or 5V.
Ease of Integration:

Simple AT Commands: Configured using AT commands for easy integration and control.
Form Factor: Available in compact sizes, making them ideal for embedded systems.

Security:

Encryption: Supports secure communication with encryption protocols.
Pairing Modes: Includes various pairing modes like PIN code, secure simple pairing (SSP).
Power Source
Bluetooth modules usually operate at a voltage of 3.3V to 5V. They are designed to work with standard power supplies available in most microcontroller and electronic development environments.

Pin Configuration
The pin configuration of a typical Bluetooth module, such as the popular HC-05 or HC-06, includes:

VCC: Connects to the power supply (3.3V or 5V).
GND: Connects to ground.
TXD: Transmits data (to the RXD pin of the microcontroller).
RXD: Receives data (from the TXD pin of the microcontroller).
State: Indicates the connection status (often not connected for simpler modules like HC-06).
EN/Key: Used to switch between command mode and data mode (varies by module).
Circuit Configuration
To use a Bluetooth module with a microcontroller, the connections are straightforward. Here's how to connect a typical Bluetooth module to an Arduino:

Power Supply Connection:

VCC: Connect to the 3.3V or 5V pin on the Arduino.
GND: Connect to the ground (GND) pin on the Arduino.
Data Connection:

TXD: Connect to the RX (digital pin 0) on the Arduino.
RXD: Connect to the TX (digital pin 1) on the Arduino (often through a voltage divider to ensure safe voltage levels).
Example Circuit Diagram
sql
Copy code

```
 Arduino
   +----------------+
   |                |
   |  [Bluetooth Module]   |
   |  VCC --- 3.3V/5V |
   |  GND --- GND   |
   |  TXD --- RX (D0) |
   |  RXD --- TX (D1, through voltage divider if needed) |
   +----------------+
```

Operation

Power Up:

When powered, the Bluetooth module initializes and is ready for pairing or communication.
Pairing:

The module can be discovered by other Bluetooth devices and paired using a PIN code or SSP.
Data Transmission:

After pairing, data can be transmitted wirelessly between the Bluetooth module and the connected device using UART communication.
Configuration:

The module can be configured using AT commands sent over the UART interface to set parameters like baud rate, name, and PIN code.
Applications
Wireless Communication:
Serial Cable Replacement: Replaces wired serial connections between devices.
Data Transfer: Used in applications requiring wireless data transfer between microcontrollers, sensors, and other peripherals.

Home Automation:

Control Systems: Enables wireless control of home appliances and lighting systems via smartphones or tablets.
Wearable Devices:

Health and Fitness: Commonly used in wearable fitness trackers and health monitoring devices to sync data with mobile applications.
Robotics:

Remote Control: Used in robotics for wireless control and data exchange between the robot and a control station.
Industrial Applications:

Machine-to-Machine Communication: Facilitates wireless communication between industrial machines and systems.

I2C MODULE

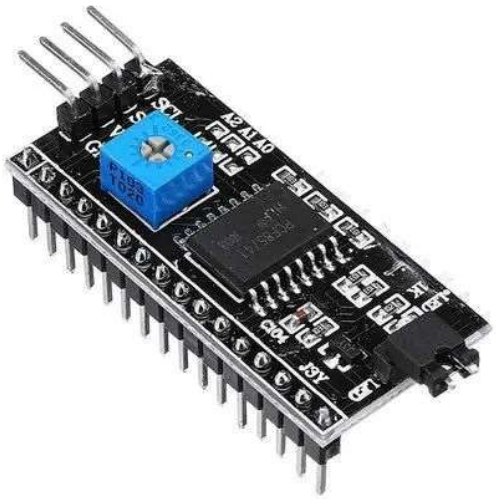


Fig 6

I2C Module: Features, Power Source, Pins, and Circuit

Features of I2C Module

1. **Two-Wire Interface**:
   - **Serial Data Line (SDA)**: Transfers data between devices.
   - **Serial Clock Line (SCL)**: Synchronizes the data transfer.
2. **Addressing**:
   - Each device on the bus has a unique address, enabling communication between multiple devices using the same two wires.
3. **Master-Slave Architecture**:
   - **Master Device**: Initiates communication and controls the clock.
   - **Slave Device**: Responds to the master's requests.
4. **Speed Modes**:
   - **Standard Mode**: Up to 100 kbit/s.
   - **Fast Mode**: Up to 400 kbit/s.
   - **Fast Mode Plus**: Up to 1 Mbit/s.
   - **High-Speed Mode**: Up to 3.4 Mbit/s.
   - **Ultra-Fast Mode**: Up to 5 Mbit/s.
5. **Simple Protocol**:
   - Communication involves start and stop conditions, address transmission, read/write bit, acknowledgment bits, and data bytes.
6. **Multimaster Support**:
   - Allows multiple master devices on the same bus.
7. **Error Handling**:
   - Includes acknowledgment checking, bus arbitration, and error detection.

Power Source

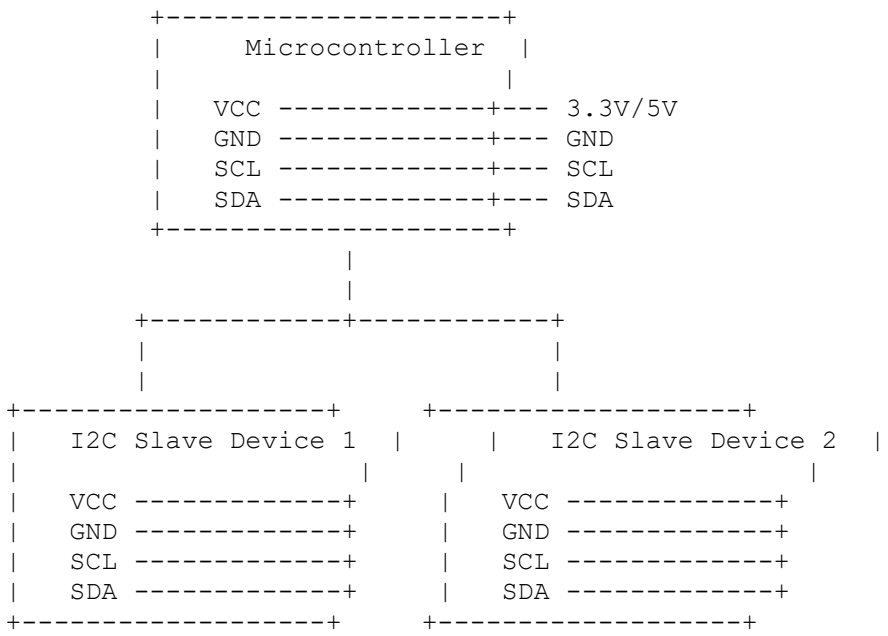
- **Voltage Range**: Typically operates at 3.3V or 5V, depending on the microcontroller and peripheral devices.
- **Current Consumption**: Low power consumption, ideal for battery-operated devices.

Pins

- **SDA (Serial Data)**:
  - Bidirectional data line used for sending and receiving data.
  - Connected to the SDA pin of both master and slave devices.
- **SCL (Serial Clock)**:
  - Clock line used to synchronize data transfer.
  - Connected to the SCL pin of both master and slave devices.
- **VCC**:
  - Power supply pin, typically connected to 3.3V or 5V power source.
- **GND**:
  - Ground pin, connected to the system ground.

Circuit Diagram

Below is a simple circuit diagram showing the connection of an I2C master and two slave devices:



In this configuration:

- The microcontroller acts as the master device.
- Two slave devices are connected to the same I2C bus.
- Pull-up resistors (typically 4.7kΩ) are connected between the SDA/SCL lines and the VCC to ensure the lines are pulled high when not driven low by any device.

Conclusion

The I2C module is a versatile communication protocol widely used in embedded systems for connecting multiple devices using just two wires. It offers various speed modes, supports multiple devices, and is easy to implement in both hardware and software, making it ideal for numerous applications in sensor communication, display modules, memory devices, and more.

LCD SCREEN

Fig 7

A 16x2 LCD screen is a popular display module used in various electronic projects and devices. It is known for its simplicity, reliability, and ease of integration with microcontrollers. The module can display 16 characters per line on 2 lines, providing a straightforward way to output text and simple symbols.

Key Features
Display Capability:

Character Display: 16 characters per line, 2 lines total.
Character Size: Typically a 5x8 pixel matrix for each character.
Backlight: Often includes a backlight for visibility in low-light conditions.
Custom Characters: Allows creation and display of up to 8 custom characters.
Controller:

HD44780: Most 16x2 LCD modules use the Hitachi HD44780 controller or a compatible equivalent, which is widely supported and well-documented.
Interface:

Parallel Interface: Supports both 4-bit and 8-bit parallel data interface modes for flexibility in pin usage.
Adjustable Contrast:

Contrast Control: Typically adjusted via a connected potentiometer.
Compatibility:

Microcontroller Support: Compatible with various microcontrollers, including Arduino, Raspberry Pi, and other development boards.
Power Source
The 16x2 LCD screen typically operates at 5V for both the logic and backlight. It can be powered directly from the 5V output of most microcontroller boards.

Pin Configuration

The standard 16x2 LCD module has 16 pins:

VSS (GND): Ground.
VDD (VCC): Power supply (5V).
VO (Contrast): Contrast adjustment, usually connected to a potentiometer.
RS (Register Select): Selects command register (0) or data register (1).
RW (Read/Write): Selects read mode (1) or write mode (0), typically connected to ground for write mode.
E (Enable): Enables the module for communication.
7-14. D0-D7 (Data Lines): 8-bit data bus (only D4-D7 are used in 4-bit mode).
A (Anode): Backlight power, typically connected to 5V through a current-limiting resistor.
K (Cathode): Backlight ground, connected to GND.
Circuit Configuration
To use a 16x2 LCD screen with an Arduino in 4-bit mode, you can connect the pins as follows:

Power and Ground:

VSS: Connect to GND.
VDD: Connect to 5V.
A: Connect to 5V through a current-limiting resistor.
K: Connect to GND.
Contrast Adjustment:

VO: Connect to the wiper of a 10kΩ potentiometer, with the other ends connected to 5V and GND.
Control Pins:

RS: Connect to a digital pin on the Arduino (e.g., D12).
RW: Connect to GND.
E: Connect to a digital pin on the Arduino (e.g., D11).
Data Pins:

D4-D7: Connect to digital pins on the Arduino (e.g., D5-D2).
Example Circuit Diagram
lua
Copy code

```
 Arduino
   +-------------------------+
   |                         |
   |   [16x2 LCD Screen]     |
   |   VSS --- GND           |
   |   VDD --- 5V            |
   |   VO --- Potentiometer  |
   |   RS --- D12            |
   |   RW --- GND            |
   |   E  --- D11            |
   |   D4 --- D5             |
   |   D5 --- D4             |
   |   D6 --- D3             |
   |   D7 --- D2             |
   |   A  --- 5V (via resistor)|
   |   K  --- GND            |
   +-------------------------+
```

Operation
Initialization:

Power the LCD and initialize it in 4-bit mode using the appropriate sequence of commands.
Display Control:

Set display parameters, such as cursor visibility and blinking, using commands.
Writing Data:

Send character data to the LCD to display text. Data is sent in two 4-bit nibbles if using 4-bit mode.
Applications
User Interfaces:

Control Panels: Display status messages and user prompts in various devices and systems.
Home Appliances: Common in microwaves, washing machines, and other consumer electronics for displaying settings and status.
Embedded Systems:

Prototyping: Widely used in microcontroller projects to provide visual feedback.
Educational Projects: Ideal for teaching how to interface displays with microcontrollers.
Instrumentation:

Measurement Devices: Display readings from sensors and instruments.
Industrial Controls: Show system parameters, alerts, and status messages.
The 16x2 LCD screen is a versatile and easy-to-use display module that is essential for many electronic projects, providing clear and concise text output. Its widespread availability and compatibility with various microcontroller platforms make it a popular choice for developers and hobbyists alike.

JUMPER WIRE -

Fig 8

Jumper wires are essential components in electronics used for making temporary connections between points on a breadboard or other prototype circuit boards without the need for soldering. They are flexible, insulated wires with connectors on each end, designed to facilitate quick and reliable electrical connections.

Features of Jumper Wires
Insulation: Jumper wires are insulated with materials like PVC or silicone to prevent short circuits and ensure safety.

Connectors: The ends of jumper wires have connectors which can be:

Male: With exposed pins that can be inserted into female headers or breadboard holes.
Female: With sockets to receive male pins from components or other wires.
Male-to-Female: Combining both types for versatile connectivity.
Variety of Lengths: Available in various lengths, typically ranging from a few centimeters to several meters, to suit different project needs.

Color-Coded: Often color-coded for easy identification of connections, aiding in the organization of complex wiring setups.

Flexibility: Made from either stranded or solid core wire, jumper wires are flexible enough to be routed through different paths on a breadboard or PCB without breaking.

Types of Jumper Wires
Male-to-Male: Used to connect points on a breadboard or to connect a breadboard to other devices or components.

Male-to-Female: Used to connect pins on a component (e.g., a microcontroller) to a breadboard or another component with female headers.

Female-to-Female: Used to connect components with male headers to each other or to a breadboard with male headers.

Applications
Breadboarding: Jumper wires are essential for creating and testing circuits on a breadboard, allowing for quick modifications and troubleshooting.
Interconnecting Components: Useful for connecting various components such as sensors, displays, and microcontrollers in a prototype.
Permanent Installations: Can be used in final installations if soldering is not preferred, especially in projects that may require frequent reconfiguration.

BREAD BOARD -

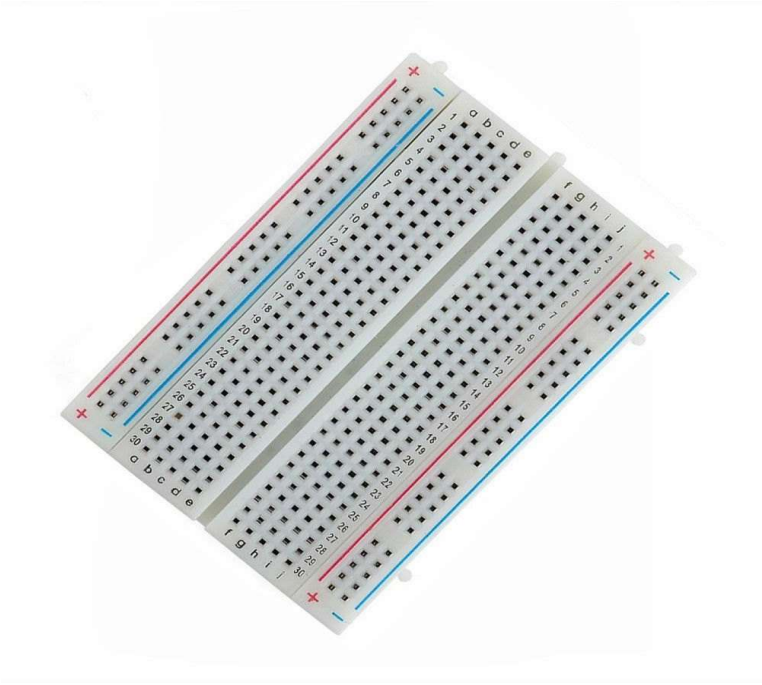


Fig 9

A breadboard is a reusable platform used for prototyping and testing electronic circuits without the need for soldering. It allows for quick assembly and modification of circuits, making it an essential tool for engineers, hobbyists, and students working on electronics projects.

Features of a Breadboard
No Soldering Required: Components and wires can be easily inserted and removed, facilitating rapid prototyping and iterative testing.

Standard Layout: Most breadboards have a similar layout with two main areas:

Terminal Strips: These are the main areas for placing electronic components. They consist of horizontal rows divided into two halves by a central gap. Each row is electrically connected.
Bus Strips: These are vertical columns along the sides, typically used for power distribution. They are marked by + and - symbols to indicate positive and negative rails.
Multiple Sizes: Breadboards come in various sizes, from small, single-component boards to large, multi-component versions, allowing for different scales of projects.

Interconnectivity: Many breadboards can be snapped together side by side to expand the working area.

Binding Posts: Some breadboards have binding posts to connect an external power supply.

Typical Breadboard Layout
A standard breadboard layout includes:

Power Rails: Two rows running along the top and bottom, often marked in red (+) and blue (-), used for distributing power.
Terminal Strips: The main area where components and wires are inserted. The terminal strips are divided by a central gap, with each half containing five connected tie points (holes) per row.
plaintext
Copy code

```
Top Power Rail (Positive)  + + + + + + + + + + + + + + + + + + + + + + + + + + + +
                           | | | | | | | | | | | | | | | | | | | | | | | | | | |
Terminal Strip (Row 1)     | | | | | |             | | | | | | | | |
Terminal Strip (Row 2)     | | | | | |             | | | | | | | | |
...                        ...                       ...
Central Gap                ------------------------------------------
Terminal Strip (Row n)     | | | | | |             | | | | | | | | |
...                        ...                       ...
```

Bottom Power Rail (Negative)- -
Using a Breadboard
To create a simple circuit on a breadboard, follow these steps:

Place the Components: Insert the leads of your components (e.g., resistors, LEDs, ICs) into the holes of the terminal strips.
Connect with Jumper Wires: Use jumper wires to make connections between components. Insert one end of the jumper wire into a hole connected to a component and the other end into the appropriate hole in another terminal strip.
Power the Breadboard: Connect the power supply to the power rails using binding posts or jumper wires.

LED-

Fig 10

An LED (Light Emitting Diode) is a semiconductor light source that emits light when an electric current flows through it. LEDs are widely used in various applications due to their efficiency, long lifespan, and versatility.

Features of LEDs
Directional Light Emission: LEDs emit light in a specific direction, reducing the need for reflectors and diffusers.

Energy Efficiency: LEDs consume less power compared to traditional incandescent bulbs, making them highly energy-efficient.

Long Lifespan: LEDs have a significantly longer operational life, often lasting tens of thousands of hours.

Variety of Colors: LEDs are available in multiple colors, including red, green, blue, yellow, and white, without the need for filters.

Fast Switching: LEDs can switch on and off rapidly, making them suitable for applications requiring fast response times.

Compact Size: The small size of LEDs allows for their use in compact and intricate designs.

Structure and Polarity
An LED typically consists of a diode made from semiconductor material, encapsulated in a plastic or epoxy case. LEDs have two leads:

Anode (positive, longer leg): Connects to the positive side of the power source.
Cathode (negative, shorter leg): Connects to the negative side of the power source.

SMPS-

An SMPS (Switched-Mode Power Supply) is an electronic power supply that uses a switching regulator to convert electrical power efficiently. The 12V 5A SMPS is a common power supply unit that provides a stable 12V DC output with a maximum current of 5 amps, making it suitable for various electronic devices and projects.

Features of a 12V 5A SMPS
High Efficiency: SMPS units are highly efficient, often exceeding 80%, as they regulate voltage by switching on and off rapidly rather than dissipating excess power as heat.

Compact Size: The use of high-frequency switching allows for smaller transformers and other components, resulting in a compact power supply design.

Stable Output Voltage: Provides a constant 12V output regardless of variations in input voltage or load conditions.

Overload Protection: Includes protection features such as overcurrent, overvoltage, and short-circuit protection to prevent damage to the power supply and connected devices.

Wide Input Voltage Range: Many SMPS units can accept a wide range of input voltages (e.g., 100V-240V AC), making them versatile for use in different regions and applications.

Applications
LED Strip Lights: Powering LED strips that require a 12V supply.
Electronics Projects: Providing power to microcontrollers, sensors, and other electronic modules.
CCTV Cameras: Supplying power to surveillance cameras and related equipment.
Home Automation Systems: Powering devices and controllers in smart home setups.
Pin Connections and Circuitry
The 12V 5A SMPS typically has the following connections:

Input Connections:

Live (L): Connects to the live wire of the AC mains supply.
Neutral (N): Connects to the neutral wire of the AC mains supply.
Earth (E or GND): Connects to the ground for safety.
Output Connections:

Positive (V+): Provides the positive 12V DC output.
Negative (V-): Provides the ground/negative connection for the 12V DC output.

BATTERY-

Fig 11

A 9V battery is a compact power source widely used in various electronic devices, including smoke detectors, transistor radios, and portable electronics. These batteries are known for their rectangular shape and versatility.

Features of 9V Batteries
Compact and Portable: The 9V battery is small and easy to carry, making it ideal for portable devices.

Standard Size: 9V batteries come in a standard size, making them easily replaceable and interchangeable across different devices.

Voltage: Provides a nominal voltage of 9 volts, suitable for many electronic circuits and devices.

Connector Type: Has a unique snap connector, with one terminal being larger than the other to ensure correct polarity when connected.

Types of 9V Batteries
Alkaline: The most common type, offering a good balance of capacity, cost, and shelf life.

Typical Capacity: Around 500-600 mAh.
Usage: Ideal for low to moderate power devices.
Lithium: Offers a higher capacity and longer shelf life compared to alkaline batteries.

Typical Capacity: Around 1200 mAh.
Usage: Suitable for high-drain devices and long-term applications.
Rechargeable: Available in NiMH (Nickel-Metal Hydride) or Li-ion (Lithium-ion) chemistries, allowing for multiple recharges.

Typical Capacity: NiMH around 200-300 mAh, Li-ion up to 600 mAh.
Usage: Ideal for devices used frequently to save on battery costs over time.
Applications
Smoke Detectors: Provides reliable power for safety devices.
Portable Radios: Powers small, handheld radios and communication devices.
Remote Controls: Used in some remote control units.
Testing Equipment: Powers multimeters and other portable testing tools.
Arduino Projects: Commonly used in DIY electronics and prototyping with microcontroller platforms.

Methodology –

Our work can be simplified into the following steps which are listed below –

- *Impact Detection:*

Force-sensitive resistors (FSRs) on the walls of the car detect impact forces.

- *Data Transmission to Arduino:*

FSRs transmit the impact force information to the Arduino board.

- *Arduino Evaluation:*

Arduino, programmed for impact assessment, determines if the force is mild or severe.

- *Visual Warning for Impact:*

Arduino sends input to a 2x16 screen via an I2C module.

1. In case of an impact within the range of 0 N to 10 N, the display shows "Mild Crash"
2. In case of an impact within the range of 10 N to 14 N, the display shows "Moderate Crash"
3. In case of an impact above 14 N, the display shows "Severe Crash"

- In all the above cases, a visual response is also generated for the driver which is implied here using blinking LED.

This system provides a real-time impact assessment, warning the driver visually as well as help send a trigger to the Bluetooth hub in future part of the project.

Circuitry involved –

- ***FSR circuit with voltage divider***

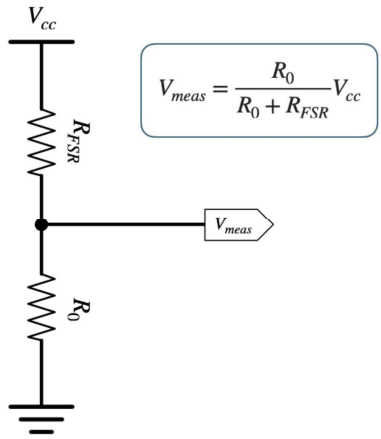


Fig 12

- ***Complete circuit diagram***

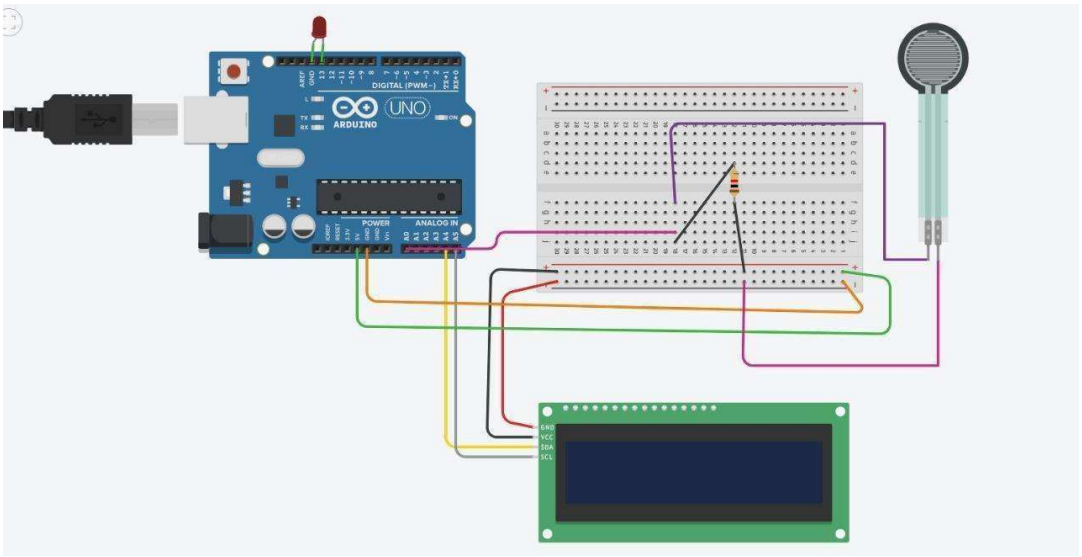


Fig 12

Connections –

Elements	Arduino pins	Breadboard points
LED	Anode – digital 13 Cathode - GND	
Breadboard	"+" 1 – 5v "-" 2 – GND	
FSR	Analog A0 – breadboard 18i	1st arm – 18f 2nd arm – "-" 11
Resistor		1st arm – "+"11 2nd arm – 18j
I2C 2x16 LCD module	SDA-A4 SCL-A5	VCC-30 "+" GND-30 "-"
9v Cell	Power Input	

SOURCE CODE OF ARDUINO IDE: -

The code input for Arduino is mentioned below –

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x27, 16, 2);
int FSR_PIN = A0;
int R0 = 1000;
int Rf = 500;
float F_known = 10.0;
float maxForce = 0.0;
String maxForceType = "None";
unsigned long lastResetTime = 0;
```

```
const unsigned long resetInterval = 15000;
int ledPin = 13;
void setup() {
  Wire.begin();
  lcd.init();
  lcd.backlight();
  lcd.begin(16, 2);
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);
}
void loop() {
  int Force_VAL = analogRead(FSR_PIN);
  float force = (Force_VAL - R0) * F_known / (Rf - R0);
  lcd.clear();
  if (force > maxForce) {
    maxForce = force;
    if (maxForce <= 10) {
      maxForceType = "Mild Crash";
    } else if (maxForce > 10 && maxForce <= 14) {
      maxForceType = "Moderate Crash";
    } else {
      maxForceType = "Severe Crash";
    }
    lastResetTime = millis();
  if (millis() - lastResetTime > resetInterval) {
    maxForceType = "None";
  }
```

```
  lcd.setCursor(0, 0);
  lcd.print("Max Force: ");
  lcd.print(maxForce, 2);
  lcd.setCursor(0, 1);
  lcd.print(maxForceType);
  Serial.println(force);
  if (maxForce > 0) {
    for (int i = 0; i < 5; i++) {
      digitalWrite(ledPin, HIGH);
      delay(500);
      digitalWrite(ledPin, LOW);
      delay(500);
    }
  } else {
    digitalWrite(ledPin, LOW);
  }
  delay(100);
}
```

Output –

Force	Output Display	LED blink
F > 0 <= 10	Mild Crash	✓
F > 10 <= 14	Moderate Crash	✓
F > 14	Severe Crash	✓

APPLICATION DATA FLOW

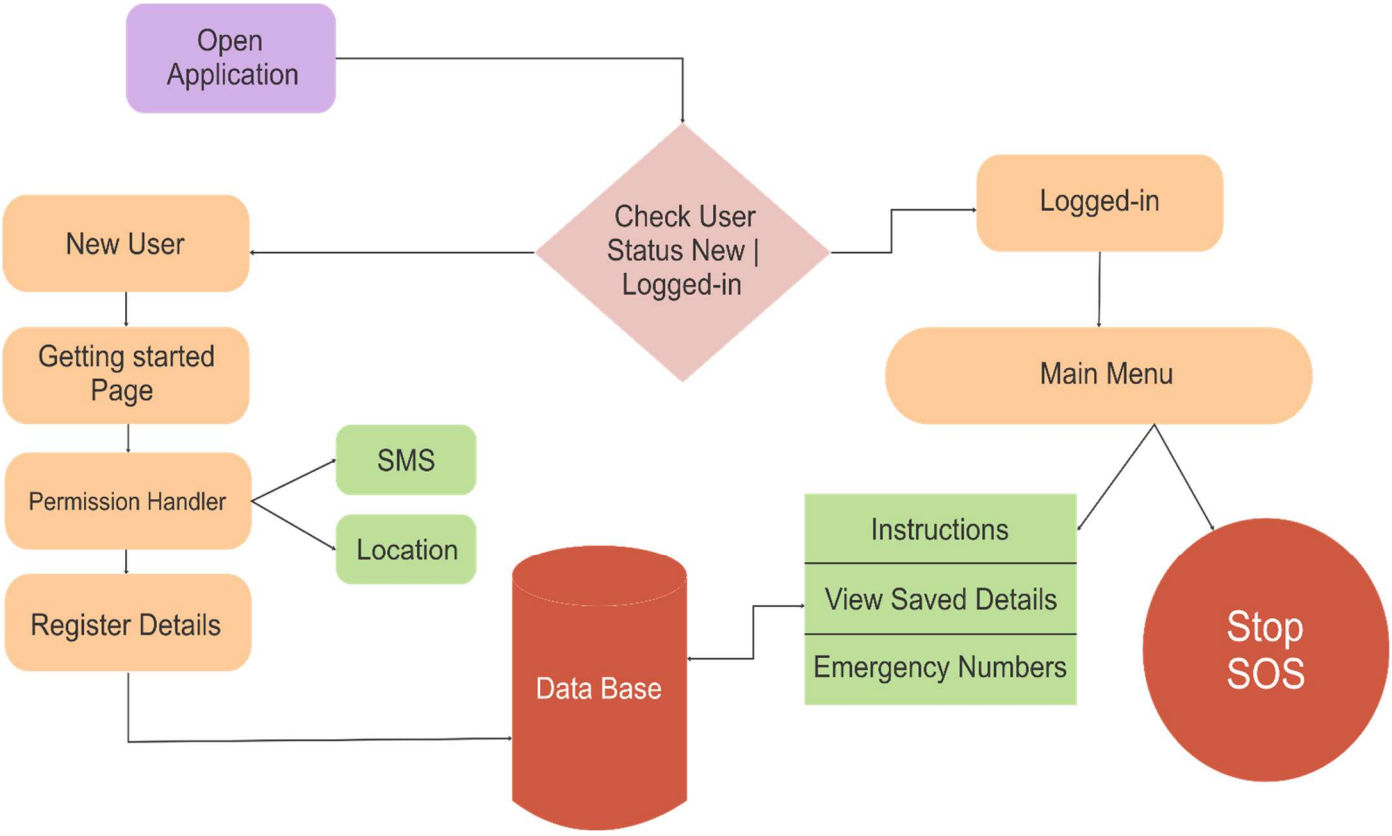


Fig 13

DESCRIPTION

The data flow in the given diagram outlines the steps a user takes when interacting with the application, along with the data management processes involved. Here is a detailed description of the data flow:

1. Open Application

- **Action**: The user opens the application.
- **Data Flow**: Initiates the application and triggers the user status check.

2. Check User Status (New | Logged-in)

- **Action**: The application checks if the user is a new user or if they have previously logged in.
- **Data Flow**:
  - If the user is new, they are directed to the "New User" path.
  - If the user is logged in, they are directed to the "Logged-in" path.

New User Path

3. New User

- **Action**: The application recognizes the user as new.
- **Data Flow**: Directs the user to the "Getting Started Page".

4. Getting Started Page

- **Action**: The user is presented with a getting started page.
- **Data Flow**:
    - The user is required to grant permissions for SMS and Location.
    - Directs the user to the "Permission Handler".

5. Permission Handler

- **Action**: The user grants necessary permissions.
- **Data Flow**:
    - Collects SMS and Location permissions.
    - Directs the user to the "Register Details" page.

6. Register Details

- **Action**: The user registers their details.
- **Data Flow**:
    - User details are collected and stored in the database.
    - Once details are registered, the user can proceed to the main menu.

Logged-in Path

7. Logged-in

- **Action**: The application recognizes the user as logged-in.
- **Data Flow**: Directs the user to the "Main Menu".

Main Menu

- **Action**: The user is presented with the main menu.
- **Data Flow**:
    - From the main menu, the user can access different options:
        - **Instructions**: View instructions for using the application.
        - **View Saved Details**: Access and view previously saved user details.
        - **Emergency Numbers**: View and manage emergency contact numbers.
        - **STOP SOS**: Access the SOS feature for emergency situations.

Database Interaction

- **Data Flow**:
    - **Register Details**: User details are saved to the database.
    - **Main Menu**: Retrieves saved user details and emergency numbers from the database.

Permissions

- **Data Flow**:
    - **SMS**: Permission is required for the application to send SMS messages.
    - **Location**: Permission is required for the application to access the user's location.

STOP SOS

- **Action**: The user can activate the SOS feature.
- **Data Flow**: Directly accessed from the main menu for emergency situations.

Summary

- The data flow starts with the user opening the application and checking their status.
- New users go through a registration and permission setup process, while logged-in users go directly to the main menu.
- From the main menu, users can navigate to various sections, including viewing instructions, saved details, emergency numbers, and the SOS feature.
- Throughout this process, user details and permissions are managed and stored in the database, ensuring data is available as needed.

This flow ensures that the application can guide new users through setup while providing quick access to essential features for returning users.

FLUTTER: -

Creating a Flutter application for emergency SOS detection involves multiple components and features to ensure users can quickly send alerts in critical situations. Here's an overview of how you can develop such an application:

Key Features of an SOS Detection App

1. **User Authentication and Registration**:
   - Allow users to sign up and log in.
   - Collect necessary details such as emergency contacts.
2. **Permissions Handling**:
   - Request permissions for accessing SMS, location, and other necessary services.
3. **SOS Activation**:
   - A button or gesture to quickly activate the SOS alert.
   - Optionally, automatic detection of emergencies based on sensors (e.g., accelerometer for fall detection).
4. **Location Tracking**:
   - Use GPS to track and send the user's location during an emergency.
5. **Communication**:
   - Send an SMS or other notifications to predefined emergency contacts with the user's location and a distress message.
   - Optionally, integrate with other communication methods like email or automated voice calls.
6. **Database Management**:
   - Store user information and emergency contacts in a secure database.
   - Manage SOS activation records for future reference.

Development Steps

1. **Set Up Flutter Environment**:
   - Install Flutter SDK and set up your development environment.
2. **Create a New Flutter Project**:

```
bash
Copy code
flutter create sos_app
cd sos_app
```

3. **Add Necessary Dependencies**:
   - Update `pubspec.yaml` with required packages like `geolocator` for location tracking, `permission_handler` for permissions, and `sms` for sending messages.

```
yaml
Copy code
dependencies:
  flutter:
    sdk: flutter
  geolocator: ^9.0.0
  permission_handler: ^10.0.0
  sms: ^0.2.4
```

4. **User Authentication**:
   - Implement user authentication using Firebase or another backend service.

- Use Firebase Auth for user sign-up and login functionalities.

5. **Permissions Handling**:
   - Use the `permission_handler` package to request necessary permissions.

```
dart
Copy code
PermissionStatus status = await Permission.location.request();
if (status.isGranted) {
  // Permission granted, proceed with location tracking
}
```

6. **Location Tracking**:
   - Use the `geolocator` package to get the current location.

```
dart
Copy code
Position position = await Geolocator.getCurrentPosition(desiredAccuracy:
LocationAccuracy.high);
```

7. **SOS Activation**:
   - Create a button or other UI element for activating the SOS alert.
   - Implement a method to send an SOS message with the current location.

```
dart
Copy code
void sendSOS() async {
  Position position = await Geolocator.getCurrentPosition(desiredAccuracy:
LocationAccuracy.high);
  String message = "Emergency! I need help. My location: ${position.latitude},
${position.longitude}";
  SmsSender sender = SmsSender();
  sender.sendSms(SmsMessage(emergencyContactNumber, message));
}
```

8. **Database Management**:
   - Use Firebase Firestore or another database to store user and emergency contact information.
   - Implement functions to save and retrieve data.

Example Code Snippet

Here is a simplified example of an SOS button implementation:

```
dart
Copy code
import 'package:flutter/material.dart';
import 'package:geolocator/geolocator.dart';
import 'package:sms/sms.dart';

void main() => runApp(SOSApp());

class SOSApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: HomeScreen(),
    );
  }
}

class HomeScreen extends StatelessWidget {
```

```
  void sendSOS() async {
    Position position = await Geolocator.getCurrentPosition(desiredAccuracy:
LocationAccuracy.high);
    String message = "Emergency! I need help. My location: ${position.latitude},
${position.longitude}";
    SmsSender sender = SmsSender();
    sender.sendSms(SmsMessage('emergencyContactNumber', message)); // Replace with
actual contact number
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('SOS App'),
      ),
      body: Center(
        child: ElevatedButton(
          onPressed: sendSOS,
          child: Text('Send SOS'),
        ),
      ),
    );
  }
}
```

Geolocator: -

The `geolocator` package in Flutter provides geolocation functionalities, allowing you to access the device's current location, handle location permissions, and perform other location-based operations. Here's an overview of how to use the `geolocator` package in a Flutter project:

Adding the Package to Your Project

First, add the `geolocator` package to your `pubspec.yaml` file:

```
yaml
Copy code
dependencies:
  geolocator: ^9.0.0
```

Then, run `flutter pub get` to install the package.

Setting Up Permissions

You need to handle location permissions for both Android and iOS.

Android

Update the `AndroidManifest.xml` file located in `android/app/src/main/AndroidManifest.xml`:

```
xml
Copy code
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.app">
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
    <application
        ...
    </application>
</manifest>
```

iOS

Update the `Info.plist` file located in `ios/Runner/Info.plist`:

```
xml
Copy code
<key>NSLocationWhenInUseUsageDescription</key>
<string>We need your location to provide better services.</string>
<key>NSLocationAlwaysUsageDescription</key>
<string>We need your location to provide better services even when the app is in the
background.</string>
<key>NSLocationAlwaysAndWhenInUseUsageDescription</key>
<string>We need your location to provide better services even when the app is in the
background.</string>
```

Using the Geolocator Package

Here is a simple example of how to get the current location using the `geolocator` package:

```
dart
Copy code
import 'package:flutter/material.dart';
import 'package:geolocator/geolocator.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: HomeScreen(),
    );
  }
}

class HomeScreen extends StatefulWidget {
  @override
  _HomeScreenState createState() => _HomeScreenState();
}

class _HomeScreenState extends State<HomeScreen> {
  String _locationMessage = "";

  void _getCurrentLocation() async {
    bool serviceEnabled;
    LocationPermission permission;

    // Test if location services are enabled.
    serviceEnabled = await Geolocator.isLocationServiceEnabled();
    if (!serviceEnabled) {
      // Location services are not enabled, don't continue
      // accessing the position and request users of the
      // App to enable the location services.
      return Future.error('Location services are disabled.');
    }

    permission = await Geolocator.checkPermission();
    if (permission == LocationPermission.denied) {
      permission = await Geolocator.requestPermission();
      if (permission == LocationPermission.denied) {
        // Permissions are denied, next time you could try
        // requesting permissions again (this is also where
        // Android's shouldShowRequestPermissionRationale
        // returned true. According to Android guidelines
        // your App should show an explanatory UI now.
        return Future.error('Location permissions are denied');
      }
    }

    if (permission == LocationPermission.deniedForever) {
      // Permissions are denied forever, handle appropriately.
      return Future.error(
          'Location permissions are permanently denied, we cannot request
permissions.');
    }

    Position position = await Geolocator.getCurrentPosition(desiredAccuracy:
LocationAccuracy.high);
    setState(() {
      _locationMessage = "Latitude: ${position.latitude}, Longitude:
${position.longitude}";
    });
  }
```

```
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Geolocator Example'),
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            Text(_locationMessage),
            ElevatedButton(
              onPressed: _getCurrentLocation,
              child: Text('Get Current Location'),
            ),
          ],
        ),
      ),
    );
  }
}
```

Key Methods in Geolocator

- **Checking Location Services and Permissions**:

  ```
  dart
  Copy code
  bool serviceEnabled = await Geolocator.isLocationServiceEnabled();
  LocationPermission permission = await Geolocator.checkPermission();
  ```

- **Requesting Permissions**:

  ```
  dart
  Copy code
  permission = await Geolocator.requestPermission();
  ```

- **Getting Current Position**:

  ```
  dart
  Copy code
  ```

- ```
  Position position = await Geolocator.getCurrentPosition(desiredAccuracy:
    LocationAccuracy.high);
  ```
  ```

Permission Handler Package: -

The `permission_handler` package in Flutter is a convenient tool for managing permissions across both Android and iOS platforms. It allows you to request permissions, check the current permission status, and open the app settings if needed.

Adding the Package to Your Project

First, include the `permission_handler` package in your `pubspec.yaml` file:

```
yaml
Copy code
dependencies:
  permission_handler: ^10.0.0
```

Run `flutter pub get` to install the package.

Setting Up Permissions

Android

Update your `AndroidManifest.xml` file to declare the necessary permissions. For example, to request location and SMS permissions, you need to add:

```
xml
Copy code
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.app">
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
    <uses-permission android:name="android.permission.SEND_SMS"/>
    <application
        ...
    </application>
</manifest>
```

iOS

Update your `Info.plist` file with the required permission descriptions. For example, for location and SMS permissions, you would add:

```
xml
Copy code
<key>NSLocationWhenInUseUsageDescription</key>
<string>We need your location to provide better services.</string>
<key>NSLocationAlwaysUsageDescription</key>
<string>We need your location to provide better services even when the app is in the
background.</string>
<key>NSLocationAlwaysAndWhenInUseUsageDescription</key>
<string>We need your location to provide better services even when the app is in the
background.</string>
<key>NSCameraUsageDescription</key>
<string>We need access to your camera for profile picture.</string>
<key>NSMicrophoneUsageDescription</key>
```

```
<string>We need access to your microphone for voice input.</string>
<key>NSSendMessageUsageDescription</key>
<string>We need access to send messages.</string>
```

Using the Permission Handler Package

Here's a simple example of how to request and check permissions using the `permission_handler` package:

```
dart
Copy code
import 'package:flutter/material.dart';
import 'package:permission_handler/permission_handler.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: HomeScreen(),
    );
  }
}

class HomeScreen extends StatefulWidget {
  @override
  _HomeScreenState createState() => _HomeScreenState();
}

class _HomeScreenState extends State<HomeScreen> {
  Future<void> _checkPermission(Permission permission) async {
    final status = await permission.status;
    if (status.isDenied) {
      if (await permission.request().isGranted) {
        print('${permission.toString()} granted');
      } else {
        print('${permission.toString()} denied');
      }
    } else if (status.isPermanentlyDenied) {
      openAppSettings();
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Permission Handler Example'),
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            ElevatedButton(
              onPressed: () => _checkPermission(Permission.location),
              child: Text('Check Location Permission'),
            ),
            ElevatedButton(
              onPressed: () => _checkPermission(Permission.sms),
              child: Text('Check SMS Permission'),
            ),
          ],
        ),
```

```
        ),
      );
    }
}
```

Explanation

1. **Checking Permission Status**:
    - The `status` method is used to check the current status of a permission.
    - Depending on the status, you can request the permission using `request()`.
2. **Requesting Permission**:
    - If the permission is denied, you can request it by calling `request()`. This will show a dialog to the user asking for the permission.
3. **Handling Permanently Denied Permissions**:
    - If a permission is permanently denied, you can prompt the user to open the app settings to manually enable the permission using `openAppSettings()`

SMS Package: -

The `sms` package in Flutter allows you to send SMS messages programmatically from within your application. This can be particularly useful for emergency apps, where sending an SOS message might be a crucial feature. Here's a step-by-step guide on how to integrate and use the `sms` package in your Flutter project.

Adding the Package to Your Project

First, add the `sms` package to your `pubspec.yaml` file:

```
yaml
Copy code
dependencies:
  sms: ^0.2.4
```

Then, run `flutter pub get` to install the package.

Setting Up Permissions

Since sending SMS requires permissions, you need to configure these in your platform-specific configuration files.

Android

Update the `AndroidManifest.xml` file located in `android/app/src/main/AndroidManifest.xml`:

```
xml
Copy code
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.app">
    <uses-permission android:name="android.permission.SEND_SMS"/>
    <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
    <application
        ...
    </application>
</manifest>
```

iOS

For iOS, permissions are handled differently and may not require explicit configuration in `Info.plist` for sending SMS using default applications. However, if you need to handle permissions, you can add the following keys to `ios/Runner/Info.plist`:

```
xml
Copy code
<key>NSLocationWhenInUseUsageDescription</key>
<string>We need your location to provide better services.</string>
<key>NSSendMessageUsageDescription</key>
<string>We need access to send messages.</string>
```

Using the SMS Package

Here's a simple example of how to send an SMS using the `sms` package:

```
dart
Copy code
import 'package:flutter/material.dart';
import 'package:sms/sms.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: HomeScreen(),
    );
  }
}

class HomeScreen extends StatelessWidget {
  final SmsSender sender = SmsSender();
  final String emergencyContactNumber = '1234567890'; // Replace with an actual contact
number

  void sendSOS() {
    SmsMessage message = SmsMessage(emergencyContactNumber, 'This is an SOS alert! I
need help.');
    message.onStateChanged.listen((state) {
      if (state == SmsMessageState.Sent) {
        print('SMS is sent!');
      } else if (state == SmsMessageState.Delivered) {
        print('SMS is delivered!');
      }
    });
    sender.sendSms(message);
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('SMS Example'),
      ),
      body: Center(
        child: ElevatedButton(
          onPressed: sendSOS,
          child: Text('Send SOS'),
        ),
      ),
    );
  }
}
```

Explanation

1. **SmsSender**: This is the object used to send SMS messages.
2. **SmsMessage**: This object represents an SMS message. You create a new instance of it with the recipient's number and the message text.
3. **onStateChanged**: This stream allows you to listen for changes in the state of the message (e.g., sent, delivered).
4. **sendSms**: This method is used to send the message.

Important Considerations

- **Permissions**: Ensure that your app requests the necessary permissions, especially on Android, as sending SMS without permissions will cause the app to crash.
- **Error Handling**: Always handle possible errors, such as permission denials or message failures, to ensure a smooth user experience.
- **Privacy and Security**: Be mindful of user privacy and secure handling of sensitive information, such as phone numbers and message content.

PROJECT PROTOTYPE –

- **HARDWARE**

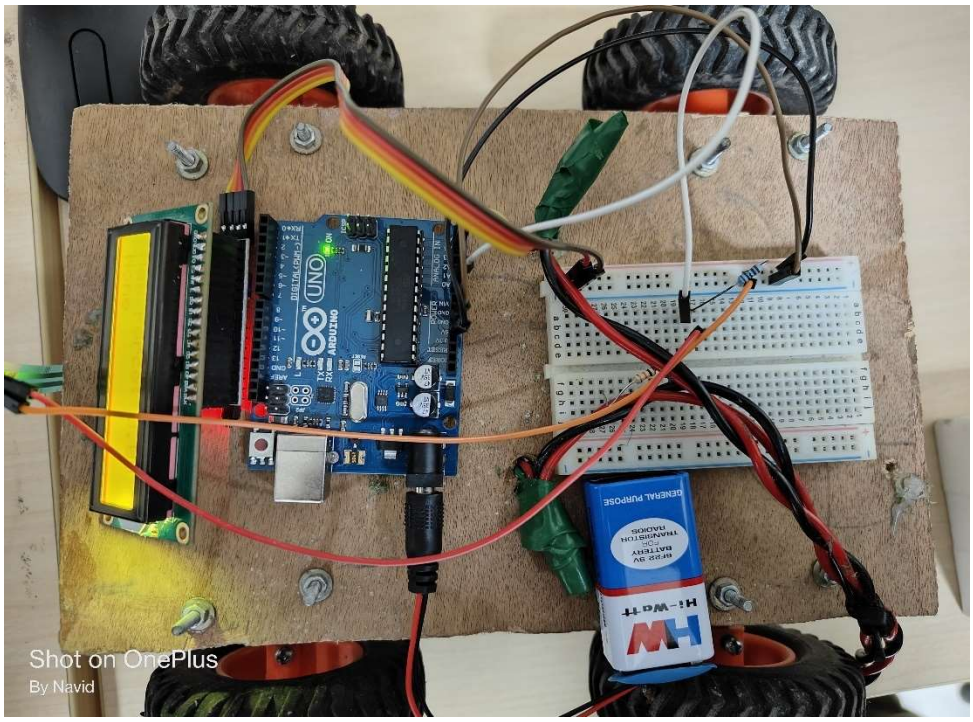
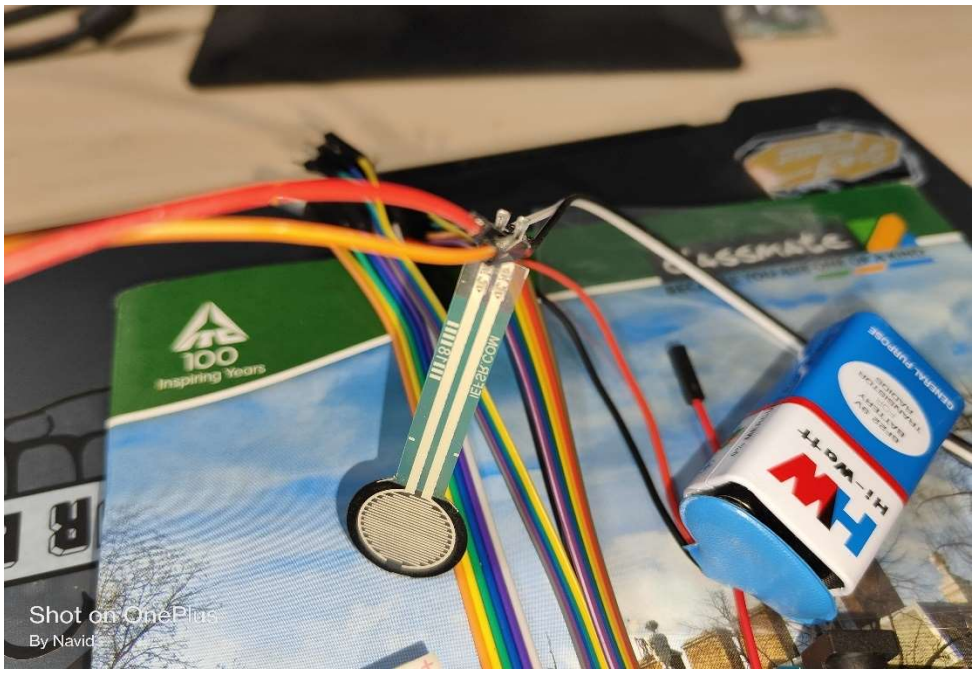


Fig 14



- **APPLICATION**

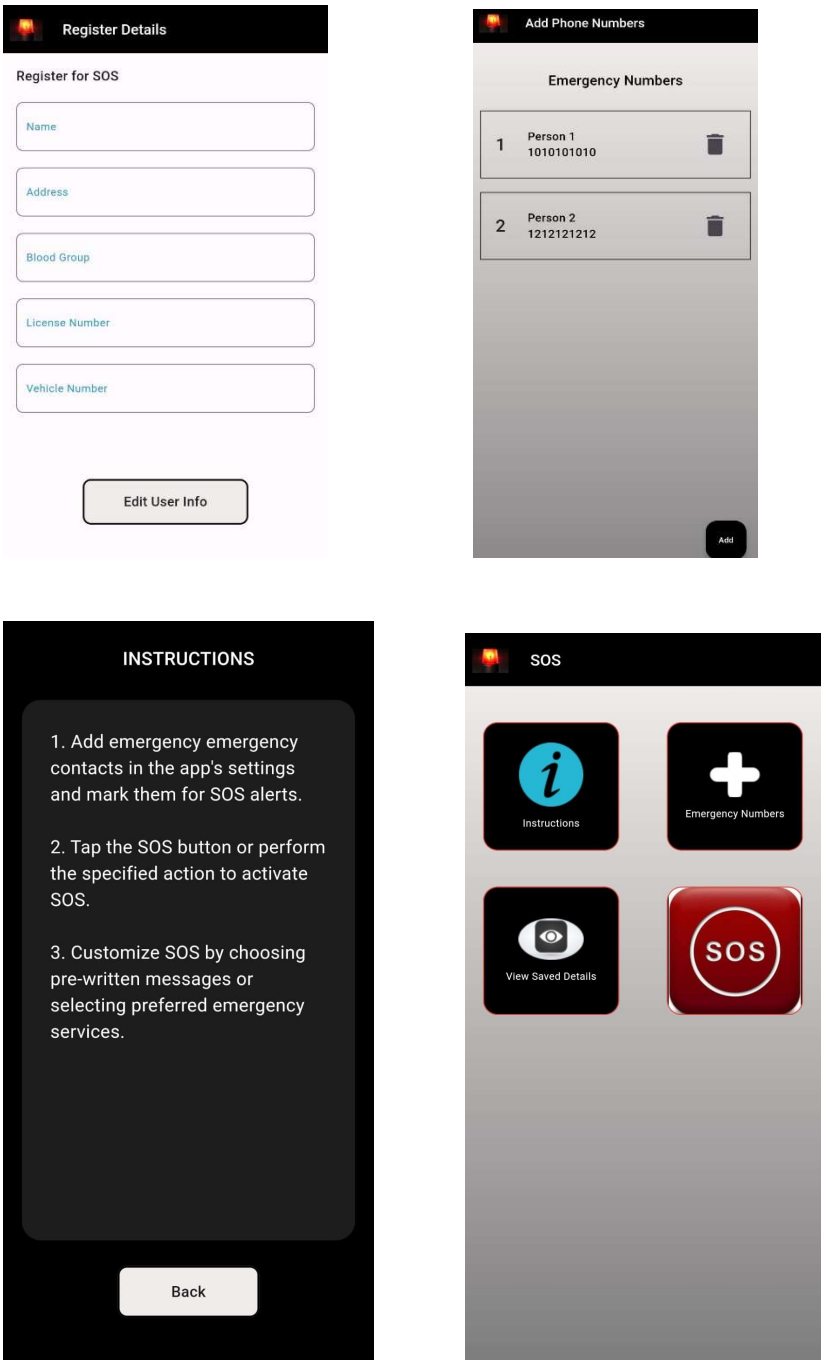


Fig 15

CHALLENGES FACED

- **UNDERSTANDING**
  The framework of the application is a complicated and maybe difficult to understand for new users.

- **TRIAL AND ERROR**
  The journey of creating this went through a lot of trial-and-error method as the codes and connections were not working properly before but after a lot of struggles everything worked out

- **APPLICATION BUILDING KNOWLEDGE**
  The application needed to be built separately to support the over all project for which we must learn a lot of new things for making it possible.

- **BUDGET**
  The main challenge was to build the proposed model in a small budget so it can be used easily and be more available in the market.

Future Aspect

In future, we are looking forward to give the whole model an IOT infrastructure by removing the present output mode and integrating it with an android device.

- We remove the Display unit and Led and connect Arduino to a Bluetooth module, which would connect it to the driver's android device.
- The mobile would have an application that would contain all vital information of the driver and vehicle.
- Upon receiving the input from Arduino, the application would compile all information into an SOS.
- The Application would take location services and determine the closest location of relief institutions and send them the SOS automatically.
- It would also have a fixed interval of time during which the driver can activate the kill switch which would stop the SOS from being sent. This is done so that unnecessary inconvenience to relief services can be prevented if the driver is fine and doesn't need help.

Conclusions

As a conclusion, the successful implementation of the progress till date marks a significant milestone in the development of an automated crash detection and notification system. The integration of force sensors, microcontrollers, and display units has demonstrated the feasibility of creating a real-time monitoring system for vehicular safety. The system accurately distinguishes between mild, moderate, and severe crashes, providing timely feedback to the driver through a combination of LED indicators and an LCD display.

Our project lays the foundation for the future expansion into more advanced stages, where the integration of IoT technology and mobile applications is anticipated. This transition promises enhanced connectivity and data utilization, allowing for more sophisticated crash analysis and immediate communication with emergency services. This forward-looking approach aligns with the evolving landscape of automotive safety systems.

The successful implementation of cost-effective components not only proves the feasibility of the concept but also underscores the potential for widespread adoption, including integration into second-hand vehicles that may lack modern safety features. The emphasis on affordability and accessibility aligns with our broader goal of ensuring vehicular safety measures are inclusive and not limited to high-end vehicles.

As our project progresses, it is expected to contribute significantly to the field of automotive safety, providing a scalable and adaptable solution for vehicles of varying specifications. The successful implementation of project progress serves as a robust foundation for future developments and underscores the importance of proactive safety measures in the automotive industry.

REFERENCE

1. Arduino:

[1] Arduino, "Arduino UNO Rev3,". Available: https://store.arduino.cc/usa/arduino-uno-rev3. 15-Mar-2022

2. Force Sensitive Resistor:

[2] Adafruit, "Force Sensitive Resistor - Square," . Available: https://www.adafruit.com/product/1075. 10-Apr-2021

3. Touch Sensor:

[3] SparkFun Electronics, "SparkFun Touch Sensor Breakout - AT42QT1010," . Available: https://www.sparkfun.com/products/12017. 20-Feb-2023

4. Battery:

[4] Battery University, "Types of Lithium-ion Batteries," . Available: https://batteryuniversity.com/article/bu-205-types-of-lithium-ion. 28-Jan-2022

5. Jump Wires:

[5] SparkFun Electronics, "Jumper Wires Premium 6" M/M Pack of 10," . Available: https://www.sparkfun.com/products/11026. 5-Dec-2021

6. Bluetooth Module:

[6] SparkFun Electronics, "Bluetooth Module - HC-05," . Available: https://www.sparkfun.com/products/12576.25-Jul-2022

7. Screen (LCD):

[7] Adafruit, "Monochrome 1.3" 128x64 OLED graphic display," . Available: https://www.adafruit.com/product/938. 30-Nov-2023
``

8. I2C Module:

[8] SparkFun Electronics, "Serial Enabled 16x2 LCD - Black on Green 3.3V," . Available: https://www.sparkfun.com/products/9395. 15-Oct-2021

9. Breadboard:

[9] Adafruit, "Half-size breadboard,". Available: https://www.adafruit.com/product/64. 20-Sep-2022

10. LED Bulbs:

[10] Adafruit, "LED - Basic Red 5mm,". Available: https://www.adafruit.com/product/300. 12-Aug-2023

11. Flutter (for mobile application development):

[11] Flutter, "Flutter Documentation," . Available: https://flutter.dev/docs. 23-May-2021

12. SMPS (Switched-Mode Power Supply):

[12] Mean Well, "Switching Power Supply Product Guide,". Available: https://www.meanwell.com/webapp/product/series.aspx?&series=RSP-200. 25-Jan-2022.