**RAJALAKSHMI ENGINEERING COLLEGE**
**RAJALAKSHMI NAGAR, THANDALAM – 602 105**


RAJALAKSHMI
ENGINEERING COLLEGE

# CB23332
# SOFTWARE ENGINEERING LAB

## Laboratory Record Note Book

Name : . . . . . .

Year / Branch / Section : . . . . . .

Register No. : . . . . . .

Semester : . . . . . .

Academic Year : . . . . . .

**RAJALAKSHMI ENGINEERING COLLEGE (AUTONOMOUS)**
**RAJALAKSHMI NAGAR, THANDALAM – 602-105**

**BONAFIDE CERTIFICATE**

**NAME:**_____ **REGISTER NO.:** _____

**ACADEMIC YEAR**: 2024-25 **SEMESTER:** III **BRANCH:** _____ B.E/B.Tech

This Certification is the bonafide record of work done by the above student in the

**CB23332-SOFTWARE ENGINEERING -** Laboratory during the year 2024 – 2025.

Signature of Faculty -in – Charge

Submitted for the Practical Examination held on _____

Internal Examiner External Examiner

# INDEX

| S. No. | Name of the Experiment | Expt. Date | Faculty Sign |
|---|---|---|---|
| 1. | Preparing Problem Statement | | |
| 2. | Software Requirement Specification (SRS) | | |
| 3. | Entity-Relational Diagram | | |
| 4. | Data Flow Diagram | | |
| 5. | Use Case Diagram | | |
| 6. | Activity Diagram | | |
| 7. | State Chart Diagram | | |
| 8. | Sequence Diagram | | |
| 9. | Collaboration Diagramt | | |
| 10. | Class Diagram | | |

## 1. PREPARING PROBLEM STATEMENT

**AIM:**

Develop an AI-powered accident detection system using sensors (accelerometers, GPS) and machine learning to identify crashes, notify emergency services, and alert nearby vehicles in real time."

**ALGORITHM:**

1. **Data Collection**: Gather real-time data from sensors (accelerometer, gyroscope, GPS) and vehicle diagnostics.
2. **Preprocessing**: Filter noise and normalize sensor data for accurate analysis.
3. **Anomaly Detection**: Identify sudden changes in speed, orientation, or impact forces using threshold or ML-based methods.
4. **Crash Validation**: Cross-check multiple data points (e.g., airbag deployment, abrupt deceleration) to confirm an accident.
5. **Alert Generation**: Trigger emergency alerts with crash details (location, severity) to emergency services and contacts.
6. **Data Logging**: Save event data for post-crash analysis and continuous system improvement.

**INPUT:**

1. **Sensor Data**: Accelerometer, gyroscope, and GPS readings for speed, orientation, and location.
2. **Vehicle Diagnostics**: Data from OBD-II (e.g., brake usage, airbag deployment).
3. **Environmental Data**: Road conditions, weather, and nearby vehicle proximity (if available).

**The Stakeholder Problem Statement**

Road accidents pose a major challenge, causing loss of life, injuries, and financial strain. Drivers often lack immediate assistance after a crash, especially in remote areas, increasing the risk of severe outcomes. Emergency responders face delays due to inaccurate or unavailable accident location data, reducing their ability to save lives effectively.

Insurance companies struggle with insufficient or unreliable accident data, leading to delays in claim processing and potential disputes. For city planners, the lack of real-time data on accident-prone areas hinders the development of safer road infrastructure.

A smart accident detection system is crucial to address these issues. It can provide real-time crash detection, accurate location tracking, and automated alerts, ensuring timely emergency response, reducing fatalities, and supporting data-driven decisions for road safety improvements.

**Problem:**

Road accidents remain a pressing issue, causing significant loss of life, injuries, and property damage. Drivers often face challenges in receiving immediate assistance after accidents, particularly in remote or isolated areas, which can worsen the situation. Emergency responders struggle with delays caused by inaccurate or incomplete accident location data, reducing their ability to act quickly and save lives.

Insurance companies face difficulties in verifying accident details, leading to inefficiencies in claim processing and frequent disputes. Additionally, city planners lack access to real-time data, limiting their ability to identify high-risk areas and implement preventive measures effectively.

The current reliance on manual reporting methods for accidents is slow and error-prone, leaving stakeholders unable to respond efficiently. A robust solution is needed to detect accidents in real time, provide accurate location data, and notify emergency services promptly. Such a system would not only improve response times but also contribute to safer roads and better data-driven decision-making.

**Background:**

Road safety has become a critical concern worldwide, with millions of accidents occurring annually, resulting in loss of life, injuries, and significant economic costs. Traditional methods of accident detection rely heavily on manual reporting, which is often delayed and prone to errors. These delays can severely impact the effectiveness of emergency response and the chances of saving lives.

The advancement of technologies like IoT, sensors, and machine learning has opened new possibilities for real-time accident detection. Modern vehicles are equipped with sensors and telematics systems capable of collecting data such as speed, acceleration, and location. By leveraging these technologies, a smart accident detection system can provide immediate alerts to emergency responders and stakeholders.

Such systems not only enhance safety by reducing response times but also offer valuable insights for city planning and insurance processing, paving the way for safer and more efficient transportation networks.

**Relevance:**

A smart accident detection system is highly relevant in addressing the growing concerns of road safety and emergency response efficiency. With increasing urbanization and vehicle density, accidents are becoming more frequent and severe. Traditional reporting methods are slow, delaying critical assistance and increasing fatalities.

By leveraging real-time data from sensors and advanced technologies like machine learning, such systems can drastically reduce response times, save lives, and improve resource allocation. Additionally, they provide accurate data for insurance claims and urban planning, making them a vital tool for modern transportation ecosystems.

**Objectives:**

- **Real-Time Detection**: Identify accidents instantly using sensor data and machine learning algorithms.
- **Accurate Alerting**: Notify emergency services and stakeholders with precise location and crash details.
- **Data Collection**: Record detailed accident data for analysis, insurance claims, and road safety improvements.
- **Response Optimization**: Reduce emergency response times to save lives and minimize accident aftermath impact.

**RESULT:**

The problem statement was written successfully by following the steps described above.

## 2. SOFTWARE REQUIREMENT SPECIFICATION

**AIM:**

The aim of the Car Accident Detector system is to detect vehicle collisions in real-time, send immediate alerts to emergency contacts, and log accident data for future analysis. This system enhances road safety by providing rapid response capabilities and storing relevant information for investigation.

**ALGORITHM:**

**a) Functionality**

Detect accidents using sensors, log data, and send alerts to emergency contacts.

**b) External Interfaces**

- **UI:** Mobile app/dashboard for user interaction.
- **Hardware:** Interfaces with vehicle sensors.
- **Communication:** Sends alerts via network (GSM, Wi-Fi).

**c) Performance**

**Speed**: Alerts within 5 seconds of detection**.**
**Availability:** Operates 24/7 with minimal downtime.
**Response**: User commands processed in 1-2 seconds**.**

**d) Attributes**

- **Functional Attributes**: Real-time accident detection, precise location tracking, and automated alerts to emergency services and stakeholders.
- **Performance Attributes**: High accuracy in detection, minimal false positives, and quick response time (under 5 seconds).
- **Security Attributes**: Encrypted data transmission and robust access control to ensure user privacy and system integrity.

**e) Design Constraints**

1. **Standards**: Follows coding standards and database normalization (3NF).
2. **Implementation Language**: Java for backend, HTML/CSS/JS for frontend.
3. **Database Integrity**: Uses primary and foreign keys for data integrity.

4. **Resource Limits**: Supports up to 100 concurrent users.
5. **Operating Environment**: Deployable on Linux/Windows, with MySQL/PostgreSQL as DB.

**SAMPLE OUTPUT:**

## 1. INTRODUCTION

### 1.1 PURPOSE

The purpose of this document is to define the requirements and specifications for the Car Accident Detector system, which aims to detect vehicle collisions in real-time and notify emergency contacts. The system is designed to enhance safety by providing immediate alerts and logging accident data for future analysis.

### 1.2 DOCUMENT CONVENTION

- **Formatting:**
  - *Italic* for emphasis
  - **Bold** for section headings
  - Monospaced for code snippets or system commands
- **Terminology:**
  - **System**: Car Accident Detector
  - **User**: Includes drivers, emergency responders, and technicians.
- **Acronyms:**
  - **SRS**: Software Requirements Specification
  - **UI**: User Interface.

### 1.3 PROJECT SCOPE

- **In-Scope:**
  - Real-time accident detection using vehicle sensors.
  - Alert generation and notification to emergency contacts.
  - Data logging of accident details for analysis.
  - Basic user interface for configuration and system status.
- **Out-of-Scope:**
  - Active collision prevention.
  - Medical assistance features.
  - Vehicle maintenance monitoring.

### 1.4 REFERENCES

- **IEEE 830-1998:** Standards for writing Software Requirements Specifications.
- **ISO/IEC 9126-1:** Software quality standards relevant to system design and implementation.
- **System Design Documentation:** For detailed architectural and design specifications.
- **User Manuals:** For instructions on system setup, configuration, and operation.

### 2.1. Performance Requirements

- **Response Time:** Detect and send alerts within 5 seconds.
- **Accuracy:** Less than 5% false positives in detection.
- **Reliability:** Operate 24/7 with minimal downtime.

### 2.2. Safety Requirements

- **Fail-Safe:** Handle sensor failures without compromising safety.
- **Error Handling:** Detect and alert on critical failures.
- **Data Integrity:** Ensure accurate data processing and storage.

### 2.3. Security Requirements

- **Encryption:** Secure data during transmission and storage.
- **Access Control:** Restrict access to critical functions.
- **Authentication:** Require secure user authentication.

### 2.4. Software Quality Attributes

- **Reliability:** Consistent detection with minimal errors.
- **Usability:** Intuitive interface for easy use.
- **Maintainability:** Easy updates and maintenance.
- **Scalability:** Support additional sensors and modules.
- **Performance Efficiency:** Minimal impact on vehicle systems.

## 4)External Interface Requirements

### 4.1. User Interfaces

- **Driver Interface:**
  - **Display:** Real-time vehicle status (speed, acceleration).
  - **Alerts:** Visual and auditory notifications for accidents.
- **Emergency Services Interface:**
  - **Dashboard:** Access to alerts and accident data with mapping.
- **Insurance Company Interface:**
  - **Reporting Tools:** Access to accident reports and trends.

### 4.2. Hardware:

- **Sensors:**
  - **Accelerometers:** Measure acceleration changes.
  - **Impact Sensors:** Detect collision forces.
  - **GPS Module:** Provide location data.
- **Microcontroller:**
  - Processes sensor data in real-time.
- **Communication Modules:**
  - **Cellular Module:** For alerts to emergency services.
  - **Bluetooth/Wi-Fi:** For local communication.

### 4.3. Software

- **Embedded Software:**
  - Processes data and detects accidents.
- **Mobile Application:**
  - Displays real-time data and alerts to drivers.
- **Server-Side Software:**
  - Manages database and APIs for communication.

### 4.4. Communications

- **Protocols:**
  - **HTTP/HTTPS:** For mobile app and server communication.
  - **MQTT:** Lightweight messaging for IoT devices.
- **Alerting System:**
  - **SMS/Email Notifications:** For immediate alerts to emergency services.

## Non-Functional Requirements

### 5.1. Performance Requirements

- **Response Time:** The system should detect an accident and send alerts within 5 seconds of the incident.
- **Data Processing:** Must handle real-time data from multiple sensors with a sampling rate of at least 10 Hz.
- **Scalability:** The system should support up to 1000 vehicles simultaneously without degradation in performance.
- **Reliability:** The system should have 99.9% uptime and effectively handle hardware failures.

### 5.2. Security Requirements

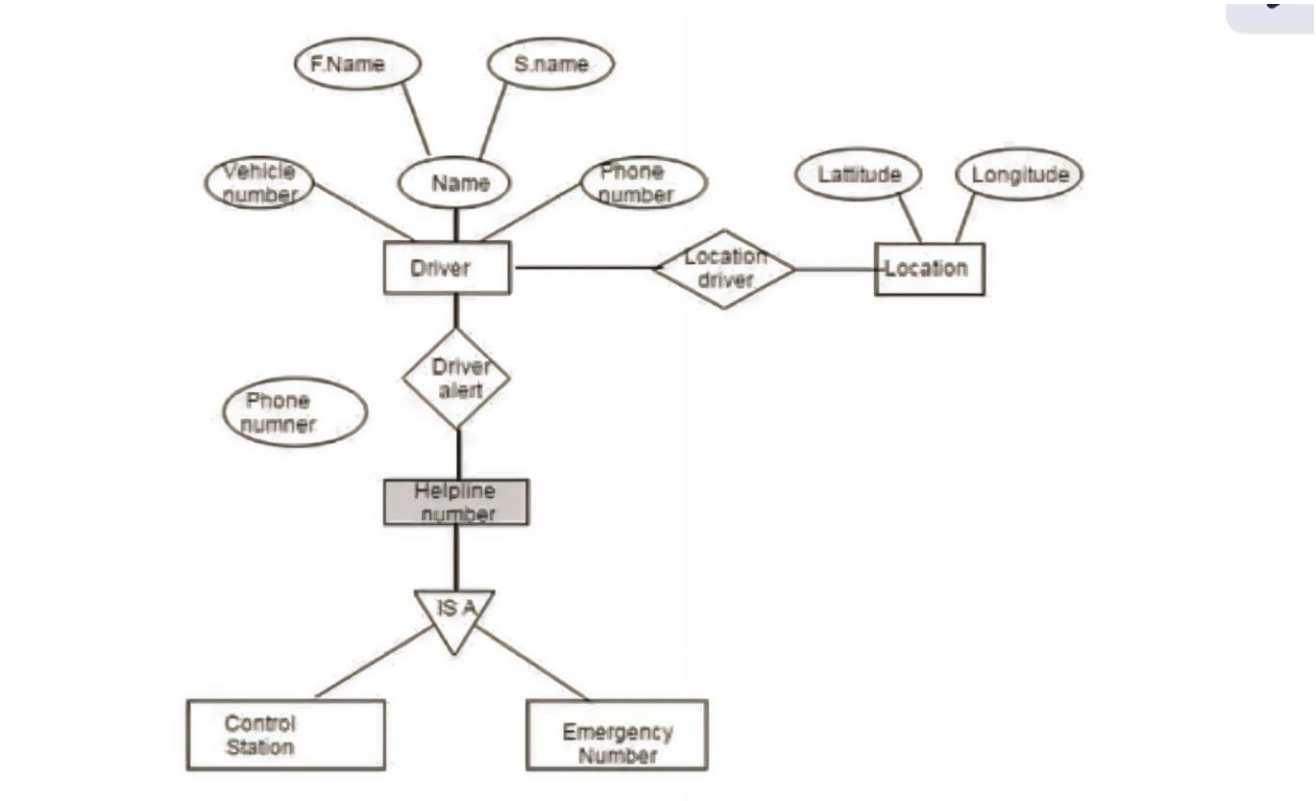
- **Data Encryption:** All data transmitted between the vehicle, mobile app, and server must be encrypted (e.g., TLS/SSL).
- **Authentication:** Users must authenticate using secure methods (e.g., OAuth for app access).
- **Data Privacy:** Personal data must be stored securely and comply with regulations (e.g., GDPR, CCPA).
- **Access Control:** Limit access to sensitive data to authorized personnel only.

### 5.3. Software Requirements

- **Platform Compatibility:** The mobile application should be compatible with iOS and Android.
- **API Standards:** RESTful APIs should be used for server communication with clear documentation.
- **Database Requirements:** Use a relational database (e.g., PostgreSQL) for storing accident reports and user data.
- **Logging:** The system should log all events (e.g., alerts sent, data processed) for auditing and debugging.

### 5.4. Safety Requirements

- **Fail-Safe Mechanisms:** Implement fail-safe features that allow the system to operate in a degraded mode if a component fails (e.g., continued monitoring).
- **Redundancy:** Critical components (e.g., sensors) must have redundancy to ensure accurate detection.
- **Error Detection:** Include mechanisms for error detection and correction in sensor data to maintain accuracy.
- **Operating Conditions:** Function effectively in a range of environmental conditions (temperature, humidity).
- **Non-Intrusive Alerts:** Alerts should minimize driver distraction and allow for manual override.



**RESULT:** The SRS was made successfully by following the steps described above.

## 3. E-R DIAGRAM

**AIM:**

To Draw the Entity Relationship Diagram Car Accident Detection.

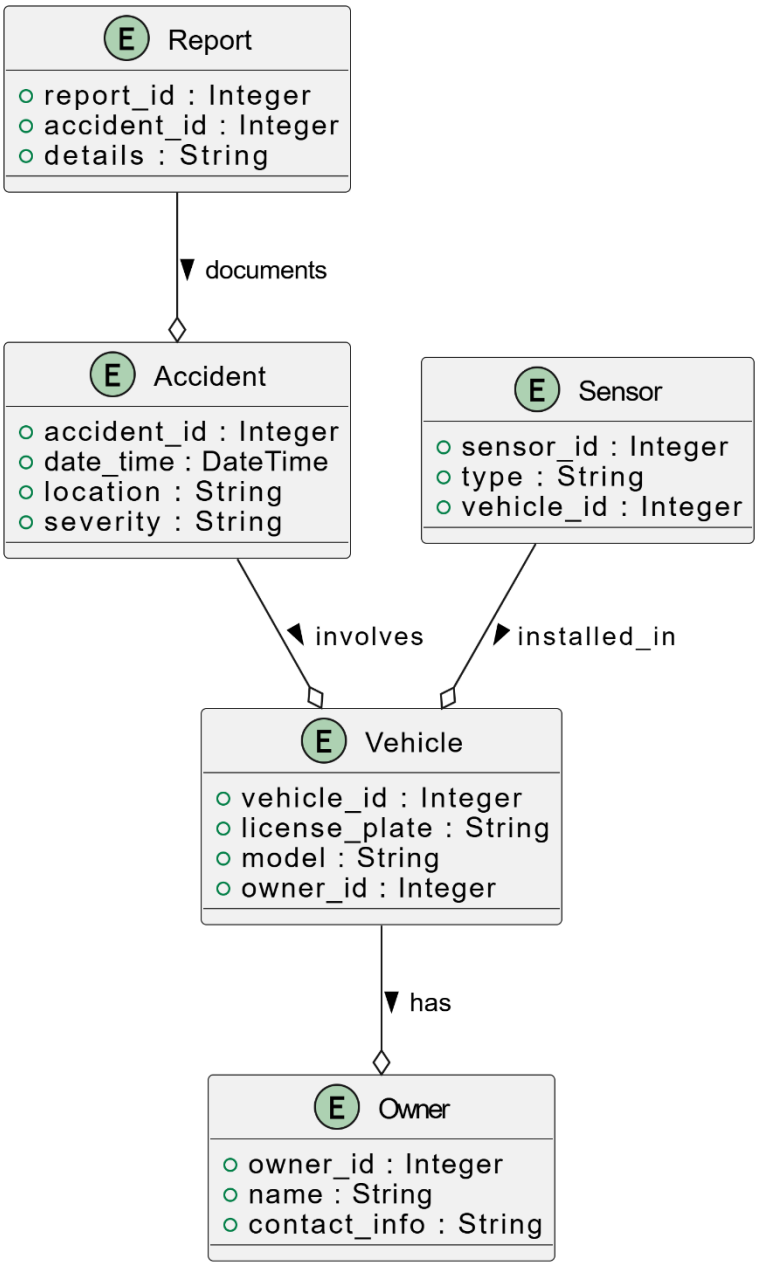
**ALGORITHM:**

1. **Identify Entities**:
   - Determine the key objects (e.g., *Vehicle*, *Accident*, *Emergency Service*, *Driver*, *Location*).
2. **Define Relationships**:
   - Establish how entities interact (e.g., *Vehicle* "reports" an *Accident*, *Accident* "notifies" *Emergency Service*).
3. **Determine Attributes**:
   - Assign relevant attributes to each entity (e.g., *Vehicle*: ID, Type; *Accident*: Time, Severity).
4. **Set Primary and Foreign Keys**:
   - Designate unique identifiers (e.g., VehicleID, AccidentID) and establish links between entities using foreign keys.
5. **Normalize Data**:
   - Eliminate redundancy by ensuring attributes are placed in the correct entity, following normalization rules.
6. **Draw the Diagram**:
   - Use standard ER diagram notations (rectangles for entities, diamonds for relationships, ovals for attributes) to create a visual representation of the system.

**INPUT:**

- Entities
- Entity Relationship matrix
- Primary Keys
- Attributes
- Mapping of Attributes with Entities

**SAMPLE OUTPUT:**




**RESULT:**

The entity relationship diagram was made successfully by following the steps described above.

# 4. DATA FLOW DIAGRAM

**AIM:**

To Draw the Data Flow Diagram for Car Accident Detection.

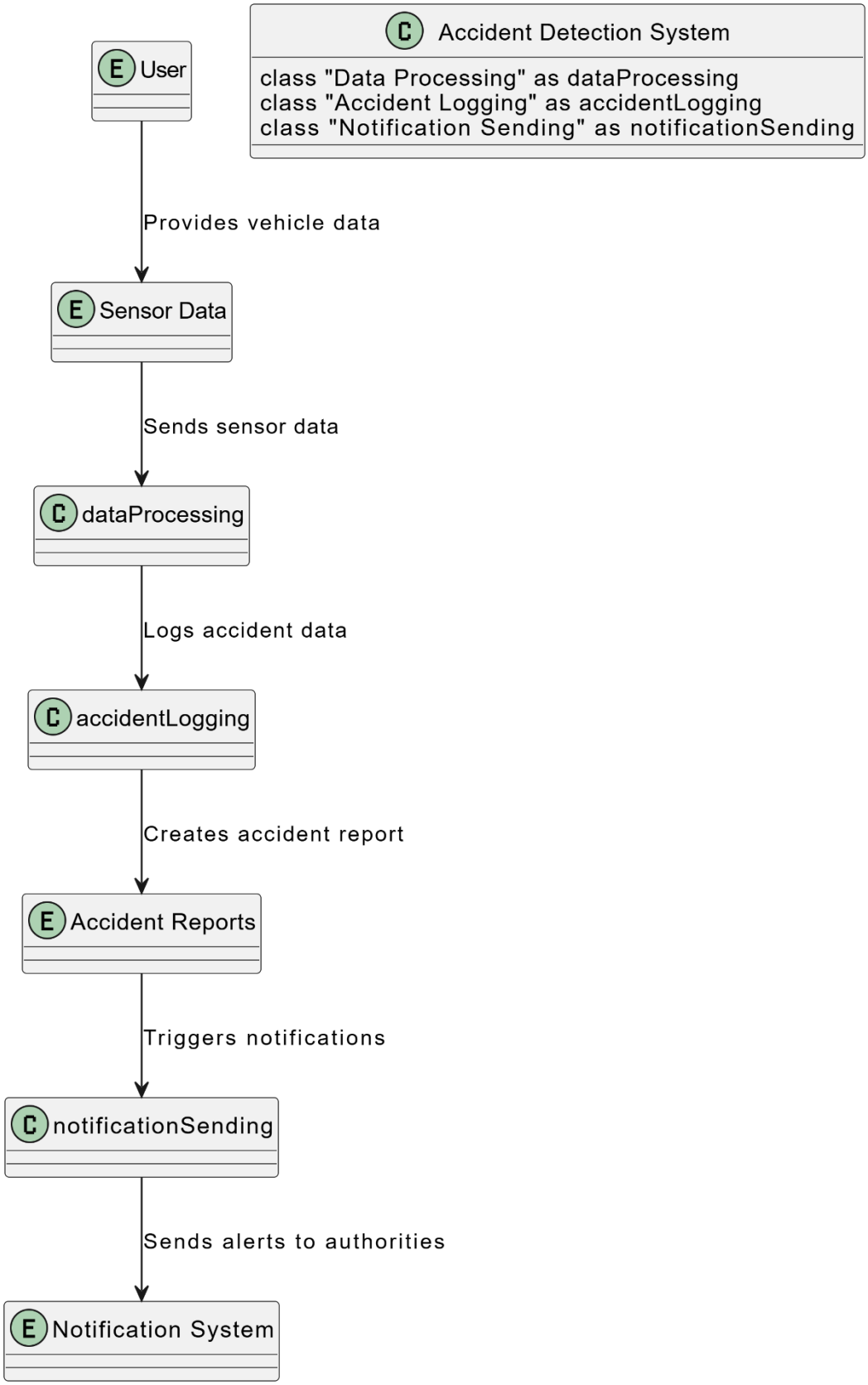
**ALGORITHM:**

- **Identify Processes**:
  - List the core functions of the system (e.g., accident detection, alert generation, data storage).
- **Define External Entities**:
  - Identify external stakeholders or systems interacting with the system (e.g., drivers, emergency services, insurance companies).
- **Determine Data Flows**:
  - Map the flow of information between processes and entities (e.g., sensor data to accident detection, alerts to emergency services).
- **Define Data Stores**:
  - Specify where data is stored within the system (e.g., accident logs, user profiles).
- **Draw the Context Diagram (Level 0)**:
  - Start with a high-level view of the system, showing external entities, the system as a single process, and the main data flows.
- **Create Detailed DFDs (Level 1, Level 2, etc.)**:
  - Break down the main process into smaller sub-processes to show detailed operations and data interactions.

**INPUT:**

- Processes
- Data Stores
- External Entities

**OUTPUT:**




**RESULT:**

The Data Flow diagram was made successfully by following the steps described above.

## 5. USE CASE DIAGRAM

**AIM:**

To Draw the Use Case Diagram for Car Accident Detection.

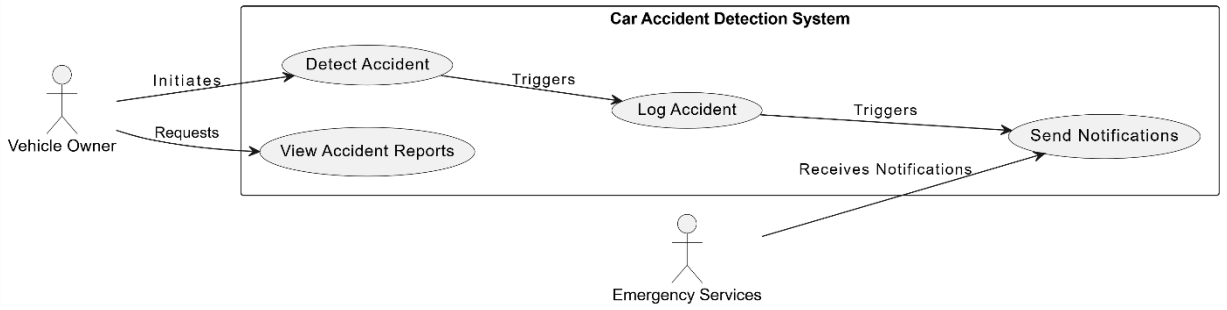
**ALGORITHM:**

- **Identify Actors**:
  - Determine the primary and secondary users or systems interacting with your system (e.g., Driver, Emergency Services, Insurer).
- **Identify Use Cases**:
  - List the actions or goals that the actors want to achieve with the system (e.g., Detect Accident, Notify Emergency Services, Generate Insurance Report).
- **Define Relationships**:
  - Establish associations between actors and use cases (e.g., Driver uses "Report Accident").
  - Include **extends** and **includes** relationships for optional or shared use cases.
- **Group Use Cases**:
  - Organize use cases logically under specific actors to ensure clarity.
- **Draw the Diagram**:
  - Use ovals for use cases, stick figures for actors, and connecting lines to represent associations.
  - Add <<extend>> or <<include>> where applicable.
- **Validate the Diagram**:
  - Ensure all actors and use cases are covered, and relationships reflect real-world interactions accurately.

**INPUTS:**

- Actors
- Use Cases
- Relations

**SAMPLE OUTPUT:**




**RESULT:**

The Use Case diagram was made successfully by following the steps given.

## 6. ACTIVITY DIAGRAM

**AIM:**

To Draw the Activity Diagram for Car Accident Detection.

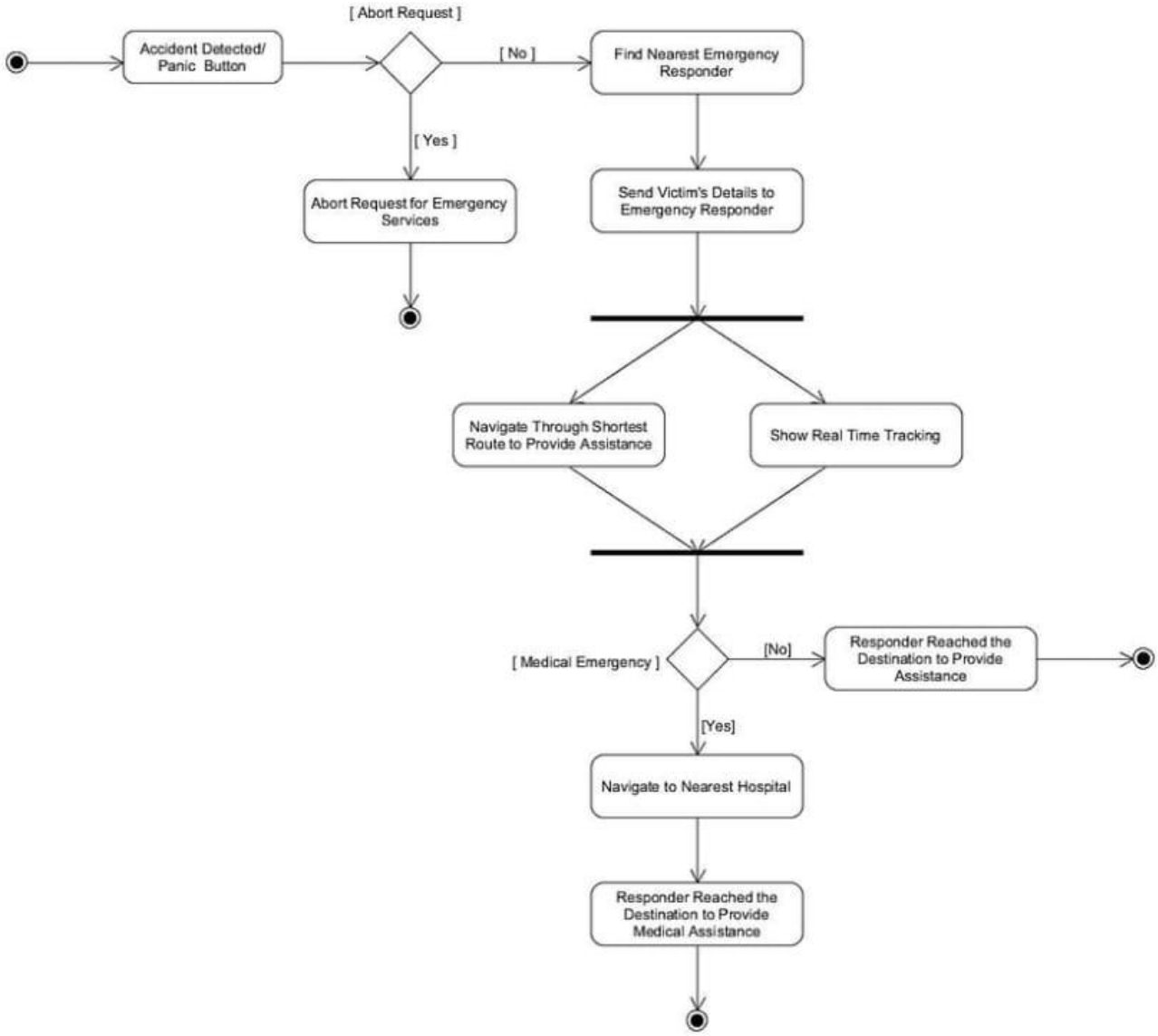
**ALGORITHM:**

- **Identify the Process**:
  - Define the workflow or process you want to represent (e.g., Accident Detection and Alert).
- **Define Activities**:
  - Break down the process into individual activities or actions (e.g., "Detect Accident," "Verify Impact," "Send Alert").
- **Determine Decision Points**:
  - Identify conditions where the process can branch into different paths (e.g., "Is it a severe accident?").
- **Sequence the Activities**:
  - Arrange the activities and decisions in the order they occur, ensuring logical flow.
- **Draw the Diagram**:
  - Use the following symbols:
    - **Ovals** for start and end points.
    - **Rectangles** for activities.
    - **Diamonds** for decision points.
    - **Arrows** to indicate the flow of control.
- **Validate the Diagram**:
  - Ensure all possible paths, loops, and end states are covered and align with the workflow's logic.

**INPUTS:**

- Activities
- Decision Points
- Guards
- Parallel Activities
- Conditions

**SAMPLE OUTPUT:**




**RESULT:**

The Activity diagram was made successfully by following the steps given.

## 7. STATE CHART DIAGRAM

**AIM:**

To Draw the State Chart Diagram Car Accident Detection.

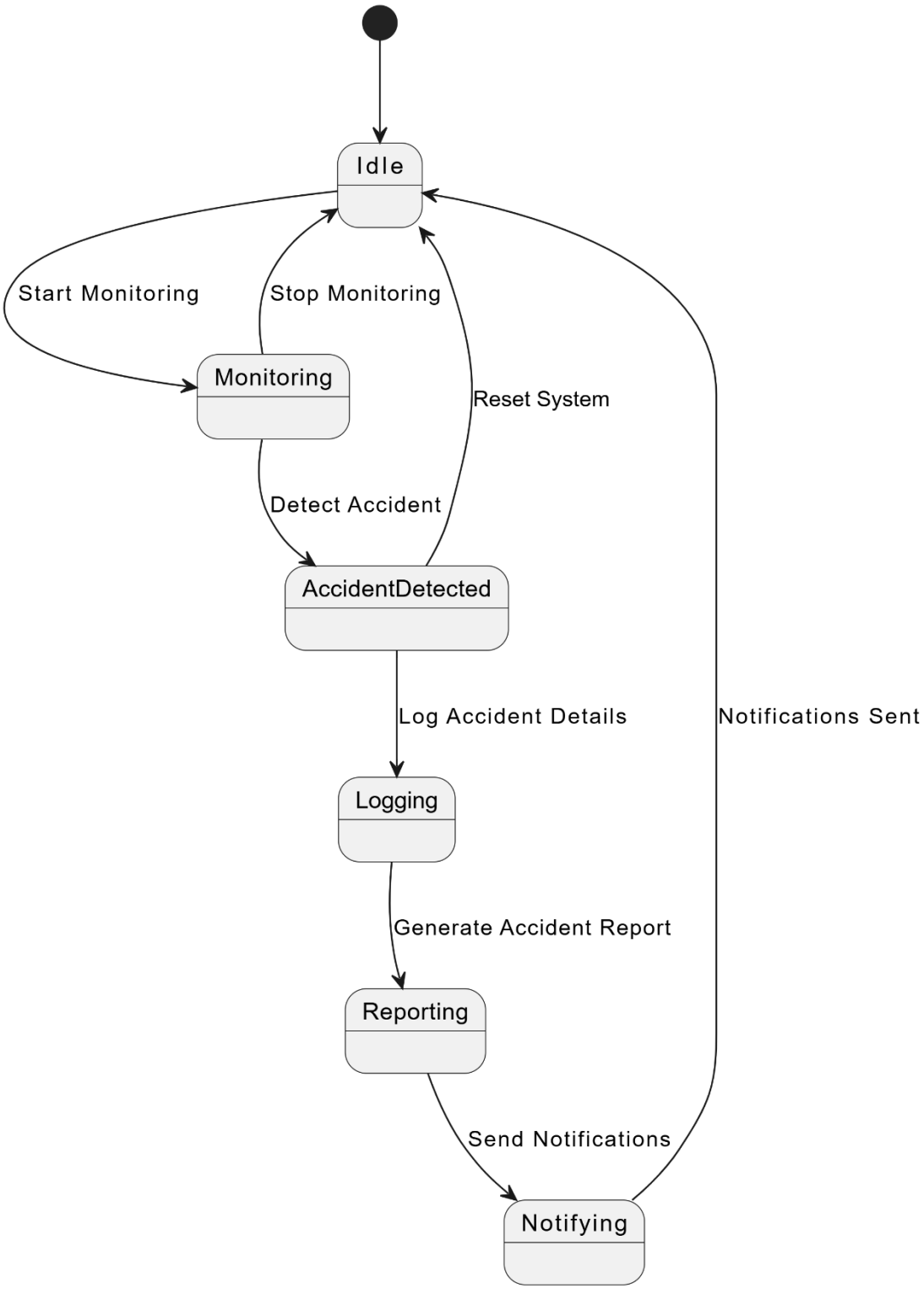
**ALGORITHM:**

- **Identify the System**:
  - Define the entity or object whose states you want to model (e.g., Accident Detection System).
- **Determine States**:
  - List all possible states the system or object can be in (e.g., "Idle," "Detecting Accident," "Alert Sent").
- **Define Transitions**:
  - Identify the events or conditions that cause the system to move from one state to another (e.g., "Impact Detected," "Alert Confirmed").
- **Add Entry and Exit Points**:
  - Mark the initial state (start) and possible final states (end) of the system.
- **Draw the Diagram**:
  - Use the following symbols:
    - **Circles** for the initial state.
    - **Rounded rectangles** for states.
    - **Arrows** for transitions between states.
    - **Bullseye circles** for the final state.
- **Validate the Diagram**:
  - Ensure all states and transitions are logical and represent the complete lifecycle of the system.

**INPUT:**

- Objects
- States
- Events

**SAMPLE OUTPUT:**



**RESULT:**

The State Chart diagram was made successfully by following the steps given.

## 8. SEQUENCE DIAGRAM

**AIM:**

To Draw the Sequence Diagram for Car Accidental Detection

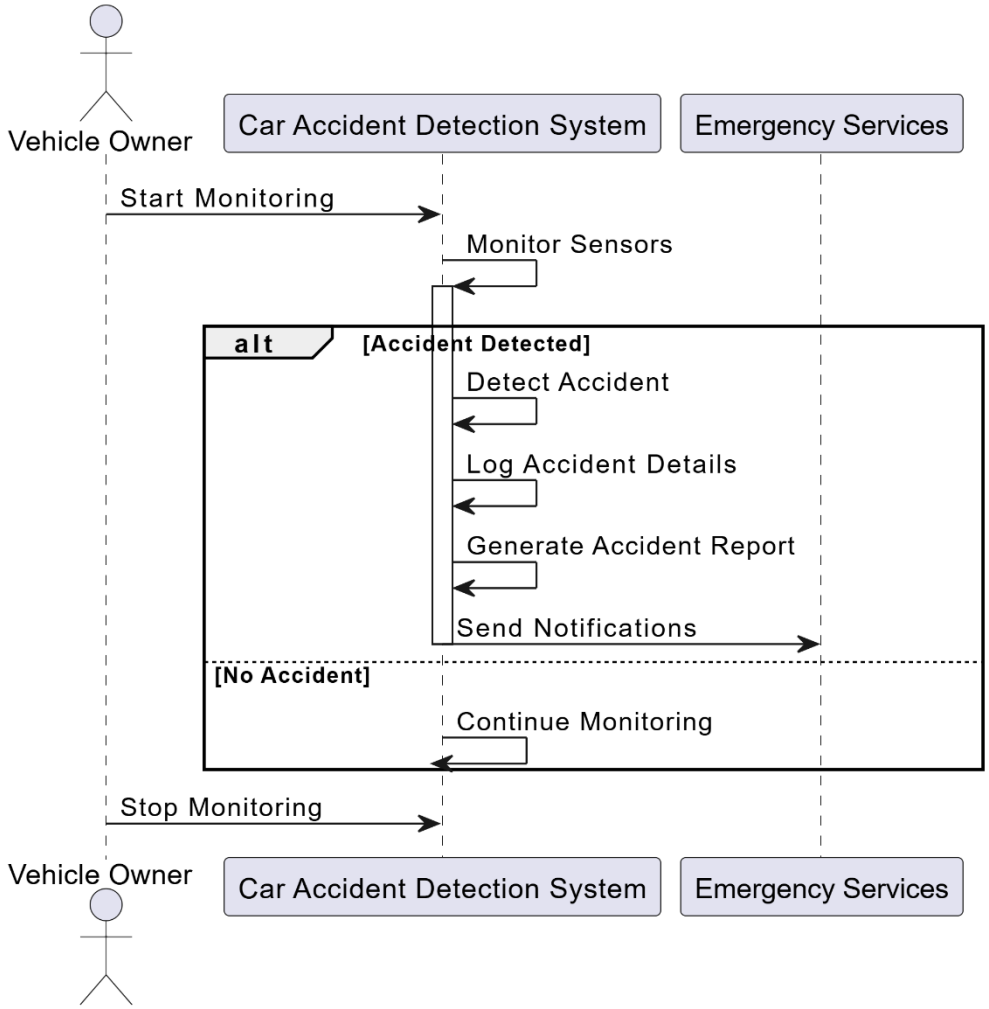
**ALGORITHM:**

- **Identify Participants**:
  - Determine the objects or actors involved in the interaction (e.g., Driver, Accident Detection System, Emergency Services).
- **Define the Interaction Flow**:
  - List the sequence of events or messages exchanged between participants (e.g., "Send Sensor Data," "Trigger Alert").
- **Draw Lifelines**:
  - Represent each participant with a vertical dashed line, starting with their name or role at the top.
- **Add Messages**:
  - Use horizontal arrows to show the messages or interactions between participants, ordered from top to bottom.
- **Include Activation Bars**:
  - Draw narrow rectangles on lifelines to indicate the active period of a participant during an interaction.
- **Handle Decision Points**:
  - Represent alternative flows or conditions using labels like “if” or “else” with separate branches.
- **Add End Points**:
  - Mark the end of the interaction with a termination symbol (an “X”) if applicable.

**INPUTS:**

- Objects taking part in the interaction.
- Message flows among the objects.
- The sequence in which the messages are flowing.
- Objects organization.

**OUTPUT:**




**RESULT:**

The Sequence diagram was made successfully by following the steps given.

## 9. COLLABORATION DIAGRAM

**AIM:**

To Draw the Collaboration Diagram for Car Accidental Detection.

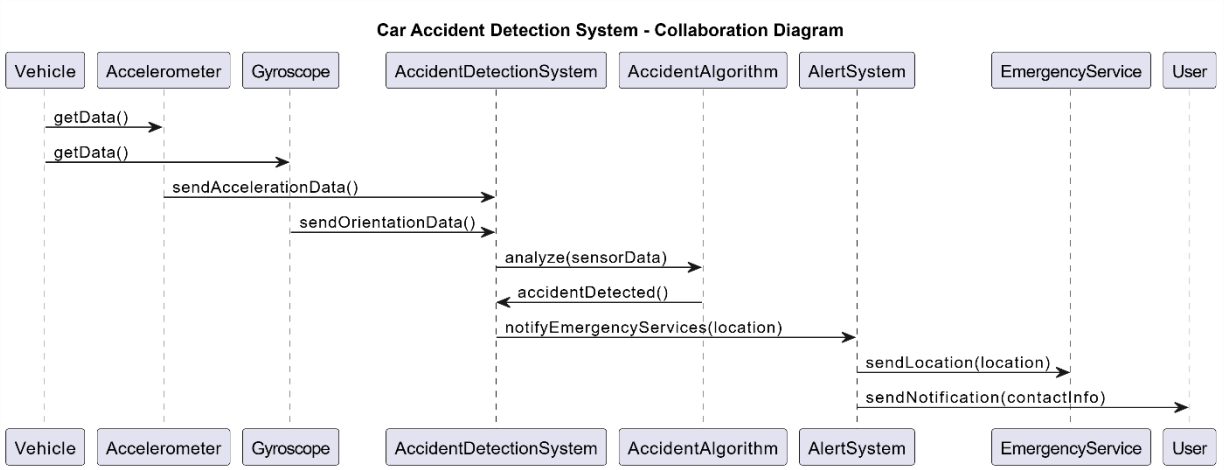
**ALGORITHM:**

- **Identify Participants**:
  - Determine the objects or actors involved in the system interaction (e.g., Vehicle, Accident Detection System, Emergency Services).
- **Define Interactions**:
  - List the messages exchanged between participants to achieve a specific functionality (e.g., "Send Data," "Notify Emergency Services").
- **Set Relationships**:
  - Establish links between participants to represent communication pathways.
- **Assign Message Numbers**:
  - Order the messages sequentially to indicate the flow of interaction (e.g., 1: Detect Accident, 2: Send Alert).
- **Draw the Diagram**:
  - Use rectangles to represent participants, and draw lines between them to denote relationships.
  - Label the lines with numbered messages showing the sequence of interactions.
- **Validate the Diagram**:
  - Ensure the messages, sequence, and relationships accurately represent the process flow and align with system logic.

**INPUTS:**

- Objects taking part in the interaction.
- Message flows among the objects.
- The sequence in which the messages are flowing.
- Object organization.

## SAMPLE OUTPUT:




## RESULT:

The Collaboration diagram was made successfully by following the steps given.

## 10. CLASS DIAGRAM

**AIM:**

To Draw the Class Diagram for Car Accident Detection.

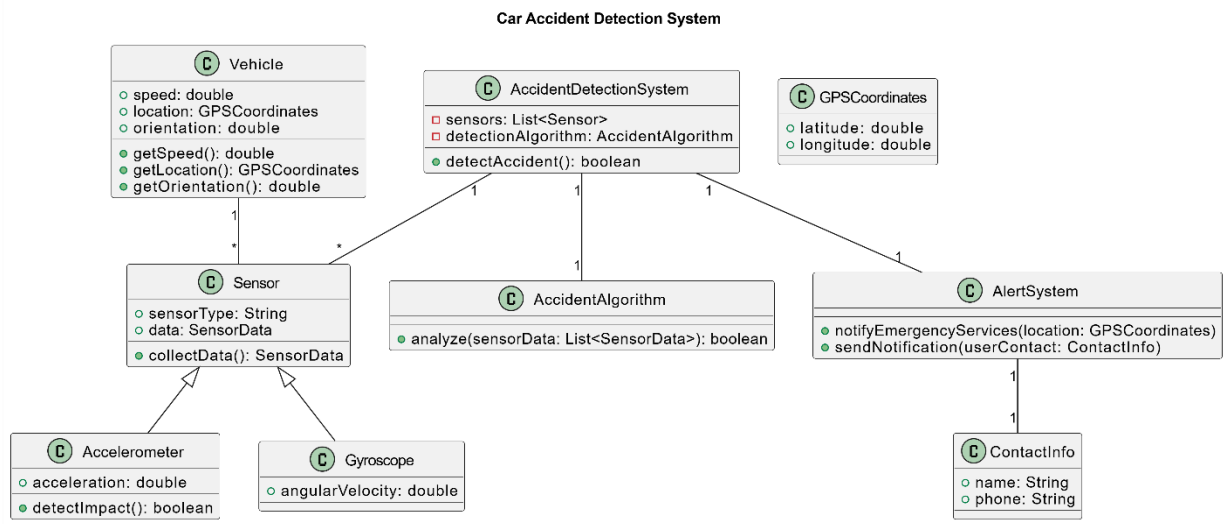
**ALGORITHM:**

- **Identify Classes**:
  - Determine the main objects or entities in the system (e.g., *Vehicle*, *Accident*, *EmergencyService*).
- **Define Attributes**:
  - List the properties or data each class will have (e.g., *Vehicle*: ID, Type, Speed).
- **Define Methods**:
  - Specify the behaviors or functions of each class (e.g., *Accident*: detectCrash(), sendAlert()).
- **Establish Relationships**:
  - Define how classes interact with each other:
    - **Association**: Simple connection between classes.
    - **Aggregation**: "Has-a" relationship (e.g., *Vehicle* has-a *Driver*).
    - **Composition**: Strong ownership (e.g., *Accident* includes *Location*).
    - **Inheritance**: "Is-a" relationship (e.g., *Car* is-a *Vehicle*).
- **Draw the Diagram**:
  - Represent classes as rectangles divided into three sections:
    - Class name.
    - Attributes.
    - Methods.
  - Use arrows to show relationships with appropriate notations (e.g., open diamonds for aggregation).
- **Validate the Diagram**:
  - Ensure all entities, attributes, and relationships are consistent and capture the system's requirements.

**INPUTS:**

- Class Name
- Attributes
- Methods
- Visibility Notation.

**SAMPLE OUTPUT:**




**RESULT:**

The Class diagram was made successfully by following the steps given.

**Code:**

**Java (using OOP concepts)**

```
 import time
import random
class Vehicle:
    def __init__(self, vehicle_id, speed, location):
        self.vehicle_id = vehicle_id
        self.speed = speed
        self.location = location
class AccidentDetectionSystem:
    def __init__(self):
        self.threshold_speed = 30  # Speed threshold in km/h
        self.accident_detected = False
    def detect_accident(self, vehicle):
        if vehicle.speed > self.threshold_speed and random.choice([True, False]):
            self.accident_detected = True
            return True
        return False
    def alert_emergency_services(self, vehicle):
        if self.accident_detected:
            print(f"Accident detected for Vehicle {vehicle.vehicle_id} at {vehicle.location}")
            print(f"Sending alert to emergency services at {time.ctime()}")
            # In a real system, GPS data and other details would be sent to emergency services
        else:
            print("No accident detected.")
class EmergencyService:
```

```
    @staticmethod
    def respond_to_alert(vehicle):
        print(f"Emergency services responding to accident for Vehicle {vehicle.vehicle_id} at {vehicle.location}.")

vehicle1 = Vehicle("V1234", 50, "12.9716° N, 77.5946° E")
accident_system = AccidentDetectionSystem()
emergency_service = EmergencyService()
if accident_system.detect_accident(vehicle1):
    accident_system.alert_emergency_services(vehicle1)
    emergency_service.respond_to_alert(vehicle1)
else:
    print("No accident detected.")
```