School of Instrumentation(SOI)

DeviAhilya VishwaVidyalaya (DAVV), Indore- 452017

IoT Based Vehicle Tracking System(VTS)

A Project Report Submitted

for the Minor Project of

Internet of Things in

School of Instrumentation

(Session 2021-2026)

SUBMITTED BY

Harshal Jaiswal

21IoT7013

DE2102078

ABSTRACT

The rapid advancements in technology and the growing need for efficient transportation management have led to the development of vehicle tracking systems. This project presents the design and implementation of a Vehicle Tracking System that leverages GPS and GSM technologies to provide real-time location monitoring. The system is also equipped with additional features, including geofencing to alert users if the vehicle moves outside a predefined area, and an emergency notification system for improved safety and security.

TABLE OF CONTENTS Page No

List of Tables

List of Figures

CHAPTER 1: INTRODUCTION

1.1.1 Overview :

The Vehicle Tracking System (VTS) is a technology-driven solution designed to monitor and manage vehicles in real-time using GPS and GSM technologies. It captures the vehicle's geographic location via a GPS module and transmits this data to a central system or user device through GSM communication. The system is equipped with a user-friendly interface that allows vehicle owners or fleet managers to track location, set geofencing boundaries, and receive alerts in case of deviations or emergencies.

The VTS servers diverse applications, including personal vehicle security, fleet managementfor logistics companies, and public transportation monitoring. By providing real-time data, it mproves operatonal efficiency, enhances vehicle security, and ensures better control over transportation systems.

1.1.2 Issues Involved

1. **Data Accuracy**: GPS signals may encounter interference from buildings, tunnels, or adverse weather conditions, impacting location precision.
2. **Network Dependency**: GSM-based communication relies on stable cellular networks, which may not always be available in remote or rural areas.
3. **Cost Constraints**: The initial setup cost for hardware components and subscription costs for GSM services can be a concern for budget-sensitive users.
4. **Privacy Concerns**: Tracking vehicles involves sensitive data, raising concerns about unauthorized access or misuse of location information.
5. **System Integration**: Ensuring seamless integration with existing software, such as fleet management tools, may require additional customization and expertise.
6. **Power Consumption**: GPS and GSM modules consume significant power, requiring careful consideration for vehicles with limited battery capacity.
7. **Scalability**: As the number of vehicles increases, managing and processing the large volume of real-time data efficiently becomes a challenge.

This project aims to address these issues while delivering a robust, efficient, and user-friendly Vehicle Tracking System.

. **1.2 Problem Definition:**

Managing vehicle security and logistics effectively is a significant challenge due to theft, unauthorized use, and the inability to track vehicles in real-time. Traditional tracking methods are unreliable and lack real-time data, leading to inefficiencies and delayed responses. There is a need for a cost-effective, scalable system that provides accurate, real-time vehicle tracking, enhances security, and improves operational efficiency. This project addresses these challenges by developing a GPS- and GSM-based Vehicle Tracking System with features like geofencing and emergency alerts for comprehensive vehicle management.

1.3 Proposed Solution:

The proposed solution is a GPS- and GSM-based Vehicle Tracking System (VTS) designed to address the challenges of vehicle security and real-time tracking. The system integrates advanced technologies with user-friendly features to deliver an efficient, scalable, and reliable solution.

The core components of the systems include a GPS module for accurate location tracking and a GSM module for transmitting this data to a central server o user devce. This enables continuous monitoring and managementof vehicles in real time. A mobile o web based interface allows users to access the system's features easily, including viewing live location data and tracking movement history.

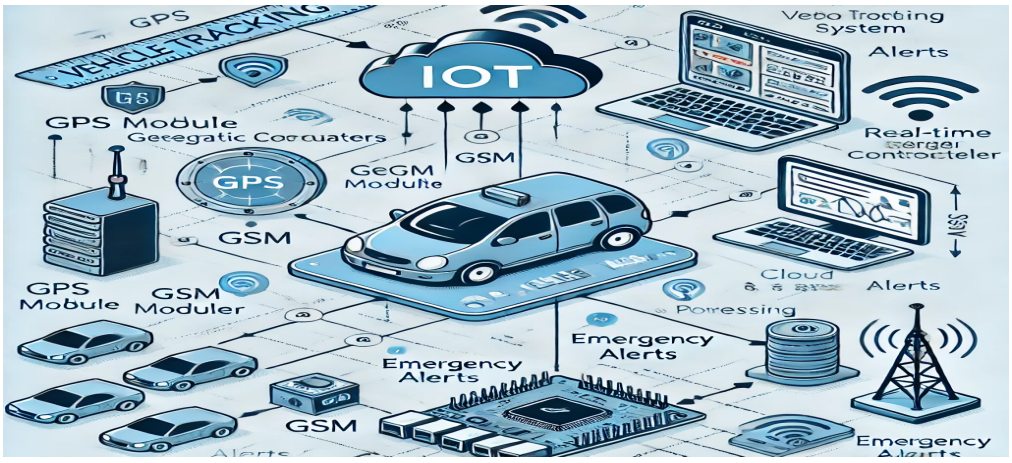


FIG. 1: IOT-ENABLED VEHICLE TRACKING

Chapter 2: Literature Survey

6.1 Review of Similar Systems

6.1.1 Comparative Analysis of Existing Systems

Several existing vehicle tracking systems have been developed, leveraging different technologies and architectures. A comparative analysis of these systems is presented below:

Table 1: Comparative Analysis between different existing systems

System Type	Technologies Used	Features	Limitations
GPS-GSM-Based Systems	GPS, GSM	Real-time location updates	Limited scalability, lacks advanced features
IoT-Based Systems	GPS, GSM, Cloud Platforms	Enhanced functionality, sensor integration	Costlier, network dependency
Mobile Application Systems	GPS, GSM	Real-time tracking via apps	UI limitations, data security concerns
Hybrid Systems	GPS, GSM, RFID	High accuracy, additional functionalities	Expensive, complex to implement

The table above provides a snapshot of different systems based on their technology, features, and limitations. Below is an expanded analysis:

1. GPS-GSM-Based Systems: These systems are widely adopted for their simplicity and affordability. They primarily use GPS for location tracking and GSM for data transmission. However, they are limited in scalability and often do not support advanced features like geofencing or emergency notifications. Their basic functionality makes them suitable for individual users but less effective for fleet management.

2. IoT-Based Systems: These systems leverage IoT to integrate GPS, GSM, and cloud platforms, enabling advanced features such as real-time analytics, geofencing, and sensor-based monitoring. They are particularly beneficial for fleet management and industrial applications. Despite their advantages, these systems rely heavily on stable network connectivity and involve higher initial costs compared to basic systems.

3. Mobile Application Systems: These solutions enhance user convenience by providing a dedicated app interface for real-time vehicle tracking and notifications. While they are

user-friendly, challenges such as limited UI/UX design, data privacy concerns, and dependency on mobile connectivity can hinder their efficiency, especially for professional use.

4. Hybrid Systems: Combining GPS, GSM, and RFID, hybrid systems deliver superior accuracy and multi-dimensional tracking. They are ideal for high-security applications and advanced fleet management. However, their complexity and high implementation costs make them less accessible for general consumers.

This comparative analysis highlights the trade-offs among existing systems, underscoring the need for a balanced solution that combines affordability, scalability, and advanced features. The proposed IoT-based system addresses these gaps by offering a cost-effective yet feature-rich alternative suitable for both personal and industrial use[1].

6.2 Critical Remarks of Previous Works

Existing vehicle tracking systems, while functional, exhibit several notable limitations and challenges. These include:

1. Dependence on Stable Network Connectivity: Many systems rely on uninterrupted GSM or internet connectivity for real-time tracking. The system's effectiveness diminishes significantly in remote or rural areas with limited network coverage.
2. High Power Consumption: GPS and GSM modules are known for their high energy consumption, making it challenging to achieve long battery life, particularly in portable or battery-operated systems.
3. Lack of Advanced Features in Basic Systems: Lower-cost solutions often lack sophisticated functionalities like geofencing, predictive analytics, or emergency alert systems, which are increasingly essential for modern use cases.
4. Security Vulnerabilities: Data transmitted over GSM or internet channels is prone to interception or hacking if not encrypted properly, raising concerns over data integrity and user privacy.
5. High Costs of Advanced Systems: Hybrid or IoT-integrated systems, while offering enhanced features, often come at a premium cost. This limits their adoption among individual users and small businesses.

These limitations underscore the need for an innovative approach to vehicle tracking that addresses these issues comprehensively. The proposed IoT-based vehicle tracking system is designed to overcome these challenges by integrating cost-efficient hardware, energy-saving mechanisms, and secure data transmission protocols. Additionally, the inclusion of advanced features such as geofencing and real-time alerts ensures a more versatile and user-friendly solution.

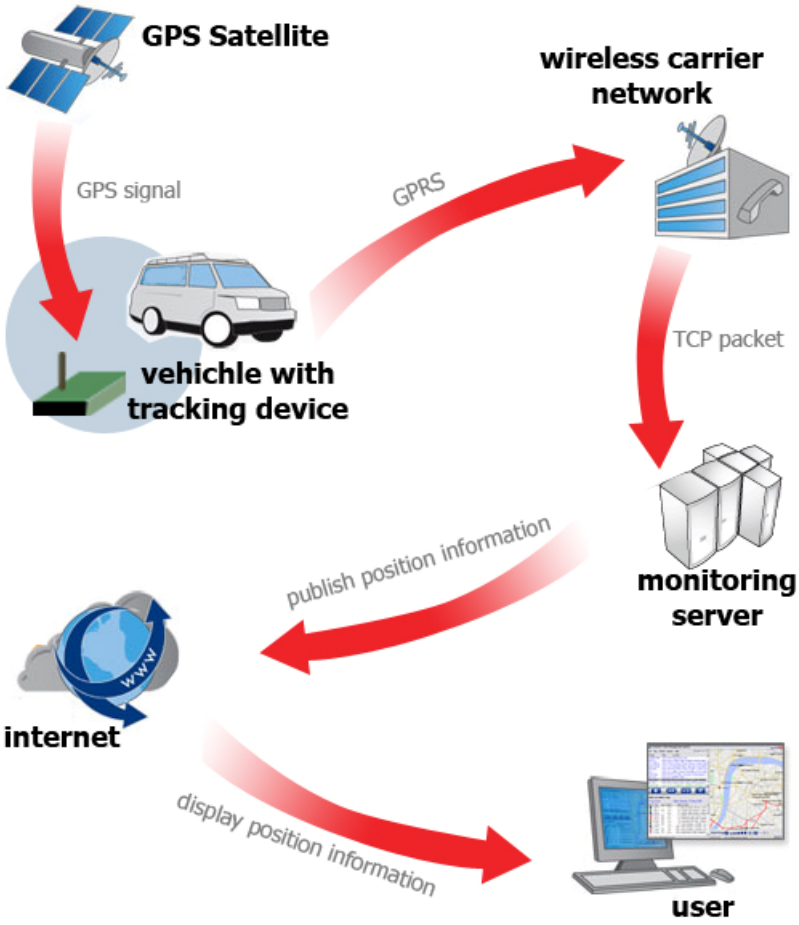


FIG. 2: TRACKING ON THE MOVE: AN IOT PERSPECTIVE

CHAPTER 3: MODULES

3.1 ESP32 Microcontroller

The **ESP32** is a low-cost, low-power microcontroller with integrated Wi-Fi and Bluetooth capabilities, developed by Espressif Systems. It is widely used in Internet of Things (IoT) projects, automation systems, and wireless communication applications due to its versatility and performance. The ESP32 is based on a dual-core processor with clock speeds up to 240 MHz, providing ample computational power for most embedded systems and IoT tasks. It also comes with a rich set of peripherals, including digital-to-analog converters (DAC), analog-to-digital converters (ADC), pulse-width modulation (PWM) pins, timers, UART, SPI, I2C, and more.
Key features of the ESP32:

- **Dual-core processor**: Operates at up to 240 MHz, which enables efficient multitasking and resource handling.
- **Wi-Fi and Bluetooth support**: Includes integrated Wi-Fi (802.11 b/g/n) and Bluetooth (classic and BLE), making it ideal for wireless communication.
- **Low power consumption**: Designed with several low-power modes, it can run for extended periods on battery-powered devices.
- **Wide I/O capabilities**: It supports multiple I/O options, making it highly adaptable for various applications.
- **Rich software ecosystem**: Compatible with popular development frameworks like the Arduino IDE, ESP-IDF, and MicroPython.

Why Choose ESP32 for the Project?

1. **Connectivity**: The ESP32 comes with integrated Wi-Fi and Bluetooth, providing out-of-the-box support for wireless communication, which is a critical requirement in many IoT projects.
2. **Processing Power**: With its dual-core processor, the ESP32 can handle multiple tasks simultaneously, making it suitable for projects that involve real-time data processing, sensor integration, or complex algorithms.
3. **Cost-Effective**: The ESP32 offers a high-performance feature set at a relatively low cost, which makes it an excellent choice for budget-conscious projects without compromising functionality.
4. **Low Power Consumption**: The ESP32's low-power modes help conserve battery life, which is particularly important for portable or remote IoT devices that need to run continuously without constant charging.
5. **Development Flexibility**: The ESP32 is compatible with popular development platforms like Arduino, ESP-IDF, and MicroPython. This flexibility ensures that developers can

choose the framework they are most comfortable with, speeding up the development process.

6. **Scalability**: The ESP32 is capable of scaling across a range of projects, from simple devices like sensors to more complex systems like home automation, wearables, or robotics, allowing it to grow with the needs of the project.
7. **Large Community Support**: With a broad user base and many active developers, the ESP32 benefits from an extensive library of resources, tutorials, and community-driven solutions, helping to resolve challenges quickly.

ESP32 Dev Board PINMAP

					3.3V	GND							
(pu)				RESET	EN	GPIO23	VSPI MOSI						SPI MOSI
SVP			ADC0		GPI 36	GPIO22							Wire SCL
SVN			ADC3		GPI 39	GPIO1	TX0						Serial TX
			ADC6		GPI 34	GPIO3	RX0						Serial RX
			ADC7		GPI 35	GPIO21							Wire SDA
		TOUCH9	ADC4		GPIO32	GND							
		TOUCH8	ADC5		GPIO33	GPIO19	VSPI MISO						SPI MISO
DAC1			ADC18		GPIO25	GPIO18	VSPI SCK						SPI SCK
DAC2			ADC19		GPIO26	GPIO5	VSPI SS					(pu)	SPI SS
		TOUCH7	ADC17		GPIO27	GPIO17							
	TMS	TOUCH6	ADC16	HSPI SCK	GPIO14	GPIO16							
(pd)	TDI	TOUCH5	ADC15	HSPI MISO	GPIO12	GPIO4		ADC10	TOUCH0		(pd)		
					GND	GPIO0	BOOT	ADC11	TOUCH1		(pu)		
	TCK	TOUCH4	ADC14	HSPI MOSI	GPIO13	GPIO2		ADC12	TOUCH2		(pd)		
				FLASH D2	GPIO9	GPIO15	HSPI SS	ADC13	TOUCH3	TDO	(pu)		
				FLASH D3	GPIO10	GPIO8	FLASH D1						
				FLASH CMD	GPIO11	GPIO7	FLASH D0						
					5V	GPIO6	FLASH SCK						

Fig. 3: ESP32 Microcontroller with pin out

3.2 GPS (Global Positining system)

GPS is a space-based satellite navigation system. It provides location and time information in all weather conditions, anywhere on or near the Earth. GPS receivers are popularly used for navigation, positioning, time dissemination, and other research purposes.

The GPS consists of satellites that orbit the earth. These satellites are geosynchronous with an orbital period that is the same as the Earth's rotation period. So they maintain exactly the same position with respect to the earth below them.

All the GPS satellites transmit radio signals, which are then captured by a GPS receiver and used to calculate their geographical position. A minimum of four satellites may be

required to compute the four dimensions of X, Y, and Z (latitude, longitude, and elevation) and time.

GPS receiver converts the received signals into position and estimates the time and some other useful information depending on the application and requirements.

GPS determines the distance between a GPS satellite and a GPS receiver by measuring the amount of time taken by a radio signal (the GPS signal) to travel from the satellite to the receiver.

o obtain accurate information, the satellites and the receiver use very accurate clocks, which are synchronized to generate the same code at exactly the same time.

If accuracy is important, you need GPS with a wide-area augmentation system (WAAS) capability. This is a satellite service providing additional correction information to the GPS receiver in order to increase its accuracy.

Before purchasing a GPS receiver, it's good to know the protocols supported by it. Some popular protocols for GPS receivers are:

NMEA 0183

An industry-standard protocol is common to marine applications defined by National Marine Electronics Association (NMEA), USA. NMEA provides direct compatibility with other NMEA-capable devices such as chart plotters and radars.

TSIP (Trimble standard interface protocol)

A binary packet protocol that allows the designer to configure and control the GPS receiver for optimal performance in any number of applications.

TAIP (Trimble ASCII interface protocol)

Designed specifically for vehicle tracking applications. It is a bidirectional protocol using simple ASCII commands with associated ASCII responses.

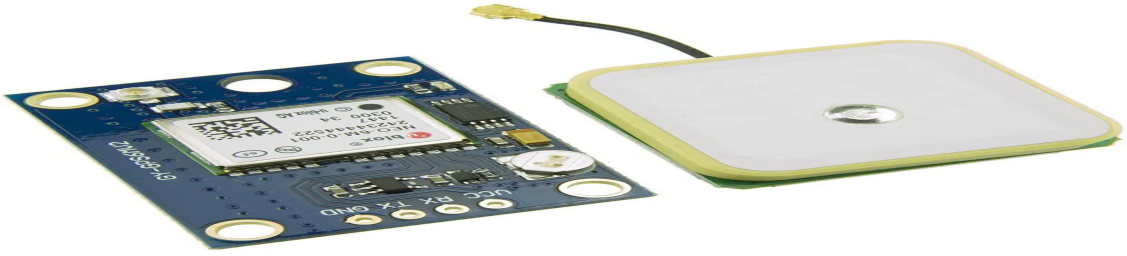


Fig. 4: GPS Module

3.3 GSM (Global System for Mobile communcation)

GSM is a standard set developed by the European Telecommunications Standards Institute (ETSI) to describe technologies for second-generation (2G) digital cellular networks.

A GSM modem is a specialized type of modem that accepts a SIM card and operates over a subscription to a mobile operator just like a mobile phone.

GSM modems are a cost-effective solution for receiving SMS messages because the sender is paying for the message delivery. To perform these tasks, a GSM modem must support an extended AT command set for sending and receiving SMS messages, as defined in the ETSI GSM 07.05 and 3GPP TS 27.005 specifications.

It should also be noted that not all phones support this modem interface for sending and receiving SMS messages, particularly most smartphones like the Blackberry, iPhone, and Windows mobile devices.

Fig. 5: GSM Module

3.4 OLED LCD

An OLED (Organic Light Emitting Diode) LCD module is a display technology that uses organic compounds to emit light when an electric current passes through them. Unlike traditional LCD screens, which rely on backlighting, OLED displays emit their own light, allowing for deeper blacks, higher contrast ratios, and more vibrant colors. OLED modules are commonly used in embedded systems, wearables, and IoT projects due to their high-quality visuals, low

power consumption, and compact size.

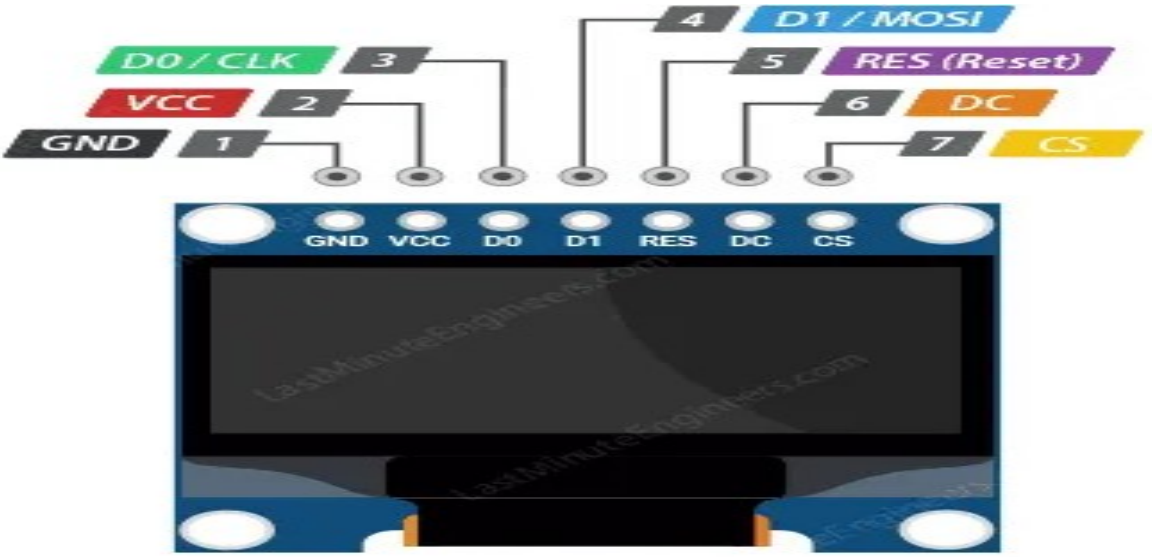


Fig. 6: OLED LCD

3.5 Blynk Software

Blynk is a popular platform for building Internet of Things (IoT) applications, allowing developers to create and manage mobile apps for controlling and monitoring IoT devices. Blynk simplifies the development of IoT systems by providing a user-friendly interface for both hardware and software components. It enables seamless communication between microcontrollers (like the ESP32) and mobile devices (smartphones or tablets), offering a cloud-based solution for real-time device management.

Key Features of Blynk:

- **Mobile App Interface**: Blynk provides a mobile app that allows users to design custom dashboards with buttons, sliders, gauges, and other widgets to control IoT devices remotely. It supports both Android and iOS devices.
- **Cloud and Local Server Options**: Blynk offers cloud-based servers for device management, as well as the option to set up a private local server for more control over data and communication.
- **Real-Time Communication**: Blynk enables real-time communication between the mobile app and connected hardware, ensuring immediate feedback and control, which is essential for many IoT applications.

- **Widgets and Customization**: The platform offers a wide range of widgets (e.g., buttons, value displays, graphs) that can be easily customized and integrated into the mobile app for intuitive device control and monitoring.
- **Device Compatibility**: Blynk supports a variety of microcontrollers (e.g., ESP32, Arduino, Raspberry Pi), making it adaptable to different hardware setups.
- **Easy Setup**: Blynk simplifies IoT device setup with a straightforward process, where users can add hardware to the platform, configure virtual pins, and connect to the mobile app without extensive coding.

CHAPTER 4: METHODOLOGY AND DESIGN

4.1 Project Flow

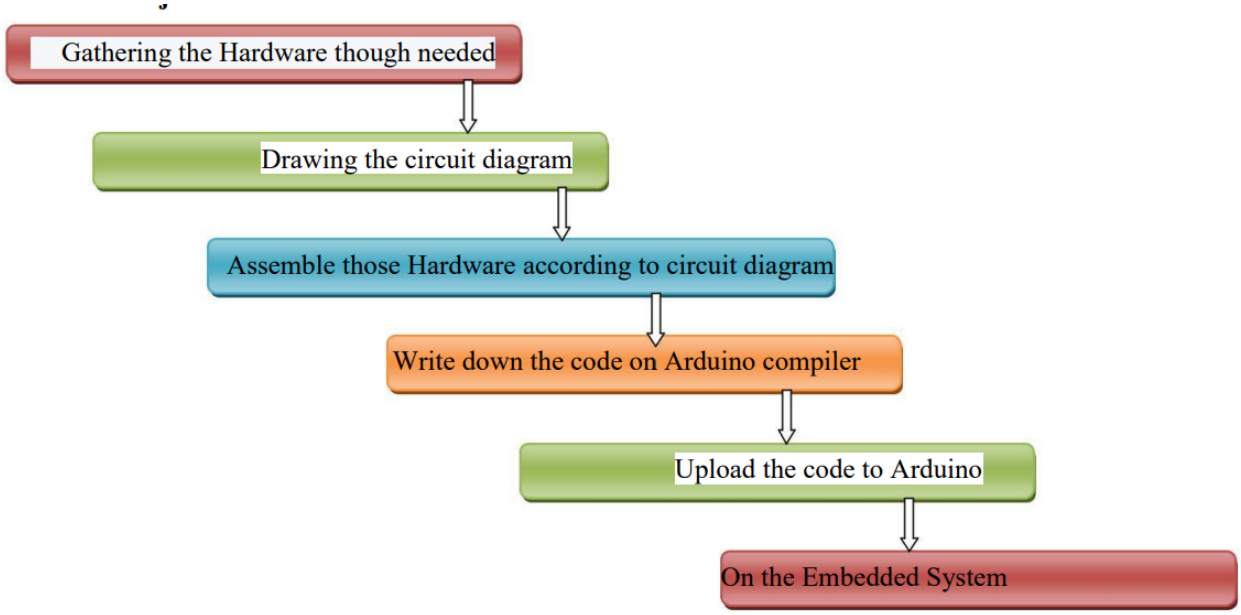


Fig. 7:Project flow

4.2 Block Diagram

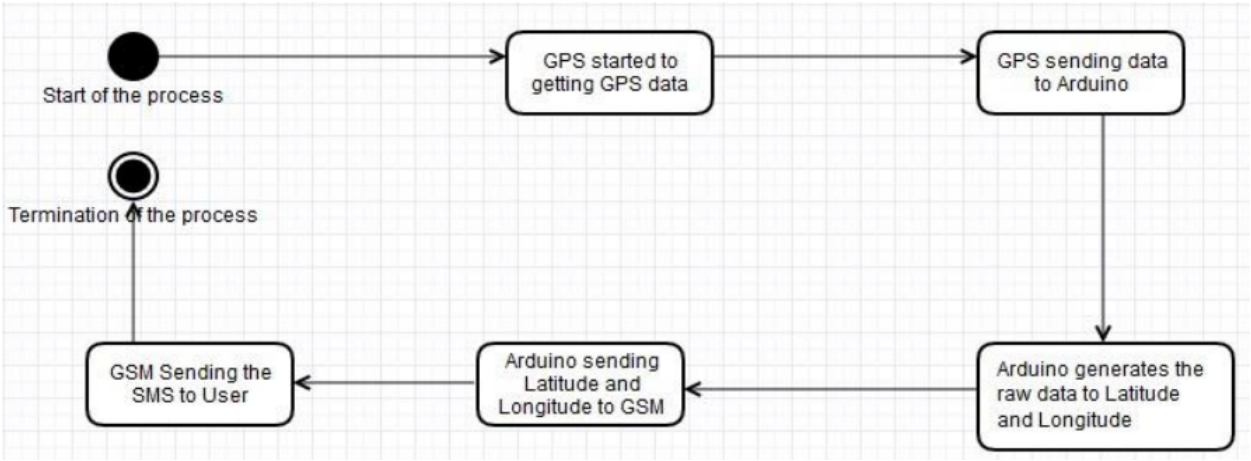


Fig. 8: Block Diagram

4.3 Working Explanation

An IoT-based vehicle tracking system consisting of an ESP32 microcontroller, GPS module, GSM module, OLED LCD, and integration with the Blynk Cloud Platform operates as follows:

1. GPS Module (Tracking Vehicle Location):

- The GPS module (e.g., NEO-6M) continuously receives signals from GPS satellites to determine the precise geographic coordinates (latitude and longitude) of the vehicle.
- The GPS module sends this location data to the ESP32 microcontroller via a serial interface (UART), allowing the ESP32 to process the information.

2. ESP32 Microcontroller (Data Processing and Communication):

- The ESP32 serves as the central control unit for the system. It receives the location data from the GPS module and processes it.
- The ESP32 is responsible for:
    - Fetching location data from the GPS module.
    - Sending this data to the GSM module for SMS alerts or to the Blynk Cloud for real-time tracking.
    - Displaying key information (such as current location or status) on the OLED LCD.
    - Connecting to Wi-Fi (if available) for sending data to the Blynk platform.

3. GSM Module (SMS Alerts and Communication):

- The GSM module (e.g., SIM800L) is used to send text messages (SMS) with vehicle location data to predefined phone numbers.
- When specific conditions are met (e.g., vehicle enters or exits a geo-fenced area), the ESP32 sends a request to the GSM module, which transmits the location data via SMS to the user's mobile phone.

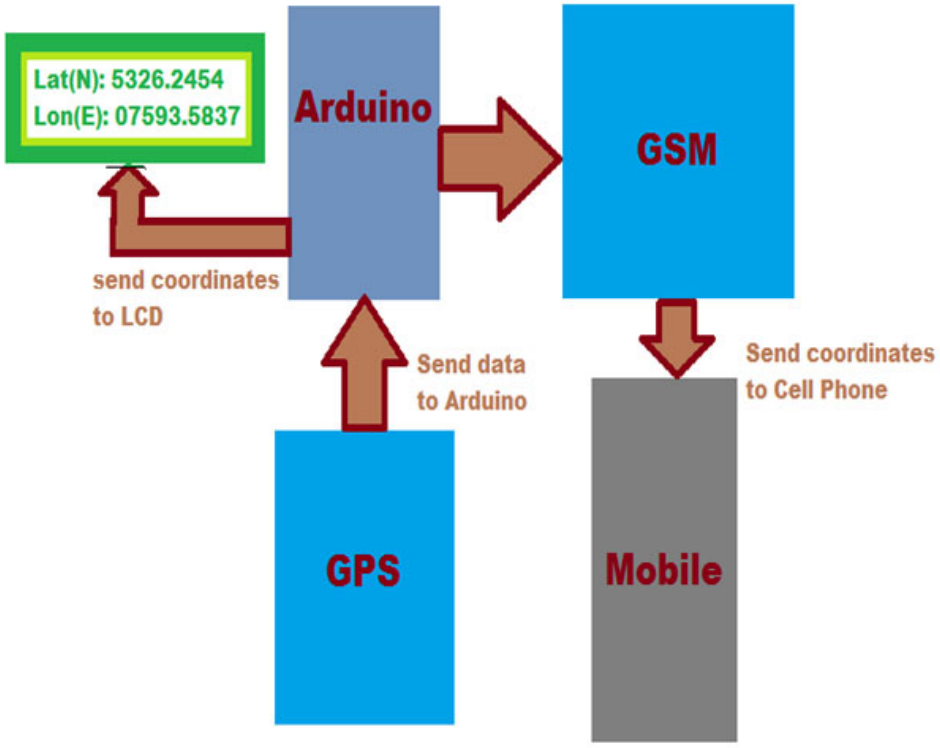
4. OLED LCD (Local Display):

- The OLED LCD (eg., 0.96-inch display) provides a real-time display of critical information such as:
    - Current latitude and longitude.
    - Speed or time of tracking.
- This allows vehicle owners or operators to quickly view the vehicle's status without needing to access a mobile app or web interface.

5. Blynk Cloud Platform (Remote Monitoring and Control):

- The ESP32 is connected to the Blynk Cloud Platform via Wi-Fi or GSM, sending real-time vehicle location and other system data.

- The Blynk mobile app allows users to monitor the vehicle's location on a map and track its movements in real-time.
    - Custom widgets on the Blynk app (e.g., GPS map, buttons, or sliders) allow remote interaction with the system.
    - Users can receive location updates, and the system can send push notifications, alerts, or updates when certain conditions are met (e.g., if the vehicle moves out of a specified area).



Fig, 9: Working Block Diagram

CHAPTER 5: ANALYSIS

The Analysis chapter provides a detailed evaluation of the methodologies, frameworks, and structures employed in developing the IoT-based vehicle tracking system. This section highlights the approaches taken during the design and implementation phases, offering insights into how the system's components interact and function.
In addition to the discussion on methodologies, this chapter may include essential subsections such as System Design and Data Flow Analysis, which further illustrate the system's architecture and operational processes. Together, these elements provide a comprehensive understanding of the system's development and functionality.

5.1 Methodologies

The methodology adopted for this project is iterative and modular, ensuring that every component of the system is independently validated. The key stages include:

1. **Requirement Analysis**:
    - Understanding the hardware and software needs.
    - Selecting components like ESP32, GPS, GSM, and compatible cloud services.
2. **System Design**:
    - Establishing communication protocols between hardware and software.
    - Designing the architecture for real-time data handling and visualization.
3. **Implementation**:
    - Coding the microcontroller with firmware for sensor integration.
    - Developing the Blynk-based mobile interface for user interaction.
4. **Testing**:
    - Performing functionality checks for real-time location tracking and geofencing.
    - Ensuring robust communication between the microcontroller and the server.
5. **Deployment**:
    - Integrating the system into a vehicle for pilot testing.
    - Analyzing performance metrics to refine the system.

5.2 System Design

This subsection elaborates on the architectural framework of the IoT-based vehicle tracking system.

- **Hardware Design**: The hardware consists of the ESP32 microcontroller, GPS module, GSM module, and other peripherals such as LCDs. Each component plays a specific role in acquiring, processing, and transmitting data.
- **Software Design**: The software integrates the hardware with cloud services using:
    - MQTT protocol for data transmission.
    - Real-time database updates for monitoring location.
    - Blynk app for user interaction.

5.2.1Flow of Data

1. **Data Acquisition**: GPS gathers real-time coordinates; sensors collect ancillary data.
2. **Processing**: The ESP32 microcontroller processes the data and encrypts it.
3. **Transmission**: GSM transmits the data to a cloud server for storage and analysis.
4. **Visualization**: Users access the data through the Blynk mobile or web interface.

5.3 Data Flow Analysis

Input Data

- The system receives:

- GPS coordinates.
- Sensor readings (optional for enhanced diagnostics).

Processing Steps

- Data is parsed, encrypted, and transmitted to the server.
- Real-time processing for geofencing alerts.

Output Data

- User receives live location updates and alerts.
- Graphical representation of historical and current tracking data.

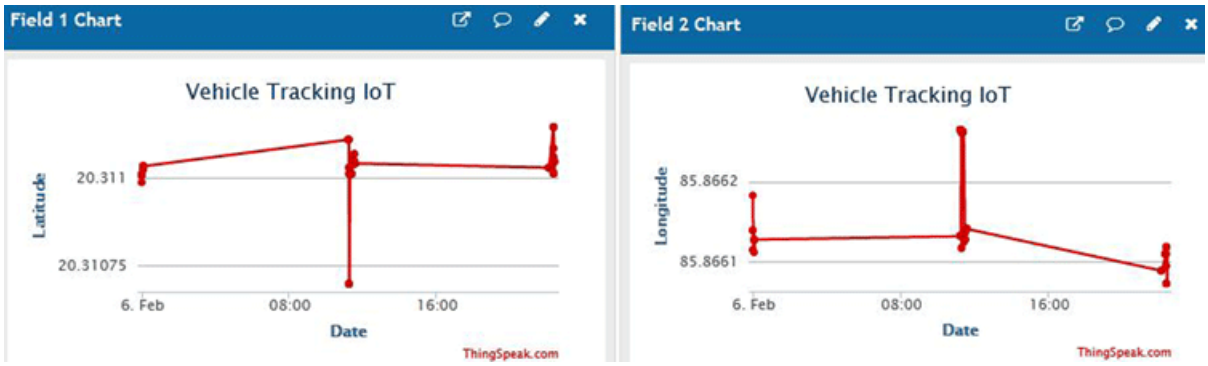


Fig. 10: Graphical Representation of coordinates(latitude and longitude)

5.4 Strengths and Limitations

Table 2: Strengths and Limitations of IOT-based Vehicle Tracking System(VTS)

Category	Strengths	Limitations
Real-Time Monitoring	- Provides continuous tracking of vehicle location.	- Dependent on stable internet/cellular network for real-time data transmission.
	- Allows fleet managers and vehicle owners to track vehicles live.	- Poor network coverage in remote areas can cause gaps in tracking or delayed updates.
Security	- Enhances vehicle security by enabling continuous monitoring.	- Vulnerable to hacking or data breaches if proper security measures are not in place.
	- Allows tracking in case of theft or unauthorized movement.	- Sensitive location and movement data could be exploited if compromised.
Cost Reduction	- Helps reduce fuel wastage through optimized route planning.	- Initial setup and installation can be costly.
	- Reduces maintenance costs by tracking vehicle conditions and maintenance schedules.	- Ongoing costs for maintenance, software updates, or repairs.
Data-Driven Insights	- Provides valuable data on driver behavior, vehicle performance, and traffic patterns.	- Reliance on data analytics may lead to incorrect conclusions if data is incomplete or misinterpreted.
	- Enables better decision-making through insights like fuel consumption and efficiency.	- Data analysis requires expertise and tools, and may involve additional costs.
Remote Monitoring	- Allows remote access to vehicle data, regardless of location.	- May not be as effective for vehicles in areas with limited connectivity or no network access.
	- Provides convenience and control, especially for fleet managers.	- Remote control features, like engine shutdown, could be misused if not properly secured.
Scalability	- Easily scalable to track multiple vehicles within a fleet.	- Complex to scale in case of network coverage limitations or integration challenges.
	- Suitable for both small businesses and large enterprises.	- Integration with other systems may become more difficult as the system grows.

Battery Life & Power Efficiency	- Can be energy-efficient with proper management.	- Some devices may have short battery life or require frequent charging if continuously active.
		- Power consumption could increase for devices requiring real-time communication (e.g., GPS).
False Alerts	- Can send instant alerts in case of unusual activity or vehicle theft.	- May generate false alerts due to signal interference, software bugs, or device malfunction.
	- Helps in quick decision-making based on alert notifications.	- False alarms could lead to unnecessary responses, wasting resources or causing confusion.
Integration with Existing Systems	- Can be integrated with fleet management and business systems for enhanced control.	- Complex integration process with existing infrastructure or third-party systems.
		- Compatibility issues may arise between different devices, software, and communication protocols.
Coverage in Remote Areas	- Provides location data even in urban areas with high network availability.	- Limited or no coverage in rural or remote areas with poor network infrastructure.
		- Inconsistent GPS signal may lead to unreliable tracking in certain environments (e.g., underground).

CHAPTER 6: IMPLEMENTATION AND TESTING

6.1 Testing

Testing Objectives

The primary objectives of testing for the IoT-based Vehicle Tracking System include:

1. Ensuring the system accurately tracks vehicles in real-time.
2. Verifying seamless communication between IoT devices and the central server.
3. Identifying and resolving defects and bugs.
4. Validating system performance under various conditions, including high traffic.
5. Ensuring compatibility across different hardware and software environments.

Testing Methodologies

Several testing methodologies were employed during the project lifecycle:

1. Unit Testing

Unit testing was conducted to verify that individual components, such as sensors and communication modules, function as expected. Automated and manual tests were used for the following:

- GPS module functionality.
- Data transmission through IoT communication protocols (e.g., MQTT).

2. Integration Testing

Integration testing ensured that different modules of the system work together seamlessly. Key focus areas included:

- Communication between GPS sensors and the IoT gateway.
- Data exchange between the IoT gateway and the cloud server.
- API communication for real-time vehicle tracking.

3. System Testing

System testing validated the complete system against specified requirements. This involved:

- Functional testing to ensure the system tracks vehicles accurately.
- Non-functional testing, including performance and scalability tests.
- Security testing to protect data against unauthorized access.

Test Cases
Test cases were designed to cover critical functionalities and edge scenarios:

Table 3: Test Cases for different modules

Test Case ID	Description	Expected Outcome	Result
TC001	Validate GPS data accuracy	Location data matches ground truth	PASS
TC002	Test data transmission over MQTT protocol	Data transmitted without packet loss	PASS
TC003	Check system response time	Response time < 2 seconds	PASS
TC004	Validate real-time tracking updates	Response time < 2 seconds	PASS
TC005	Ensure compatibility with mobile app	Consistent data display on app	PASS

6.2 Circuit diagram

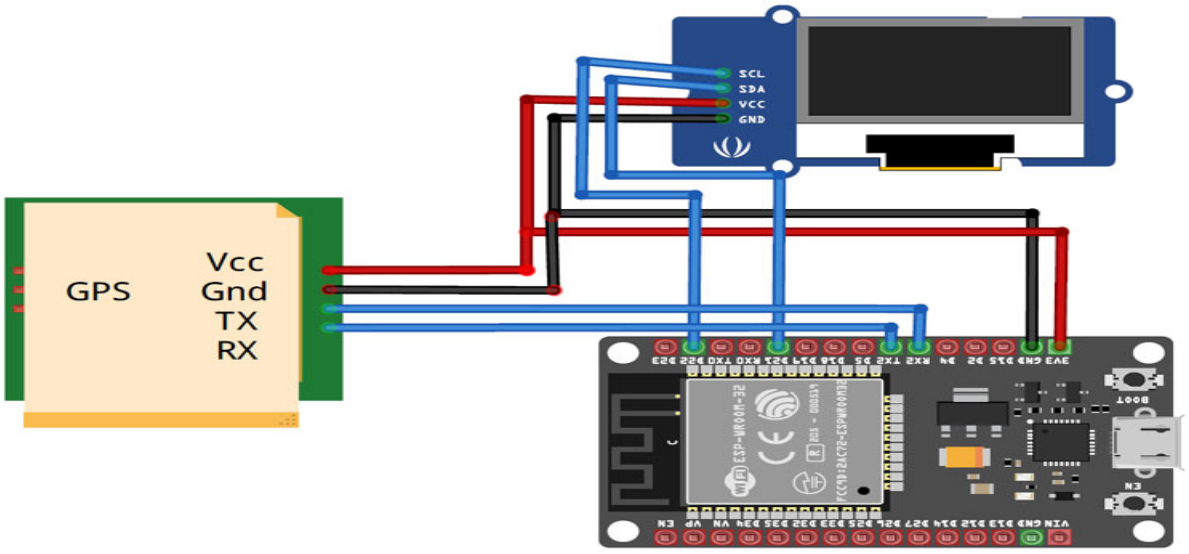


Fig. 10: Circuit Diagram

The provided circuit diagram represents the hardware configuration of an IoT-based vehicle tracking system. Here's a detailed explanation of the connections:
Components:

1. GPS Module: Provides real-time location data.
    - Vcc: Power input, connected to the 3.3V pin of the ESP32 microcontroller.

- GND: Ground, connected to the GND pin of the ESP32.
- TX (Transmit): Transmits GPS data to the microcontroller, connected to the RX (Receive) pin of the ESP32.
- RX (Receive): Not used in this setup.

2. ESP32 Microcontroller: Serves as the central processing unit, managing communication between the GPS module and the display.
    - 3.3V Pin: Supplies power to the GPS module and OLED display.
    - GND Pin: Provides a common ground to all components.
    - I2C Pins: Manages communication with the OLED display:
        - SDA (Data Line): Connected to the SDA pin of the OLED display.
        - SCL (Clock Line): Connected to the SCL pin of the OLED display.
3. OLED Display: Displays real-time tracking information.
    - SCL (Clock Line): Connected to the SCL pin of the ESP32 for I2C communication.
    - SDA (Data Line): Connected to the SDA pin of the ESP32.
    - Vcc: Power input, connected to the 3.3V pin of the ESP32.
    - GND: Ground, connected to the GND pin of the ESP32.

6.3 Code Snippets

- The ssid and pass variables are used by the device (e.g., ESP32 or ESP8266) to establish a connection to the specified Wi-Fi network.
- The auth variable is used to authenticate the IoT device with the Blynk cloud service, enabling features like remote control, data visualization, and notifications.

```
const char *ssid = "Poco x3"; // Enter your Wi-Fi Name
const char *pass = "rrrrrrrr"; // Enter your Wi-Fi Password
char auth[] = "76aeba5832304e10917b4e1748c34039";
```

The code checks for available GPS data in SerialGPS.

1. It parses the data using the GPS library and verifies its validity.
2. If valid, it extracts the latitude and longitude, converts them to strings with six decimal places, and stores them in lat_str and lng_str.

This snippet is crucial for retrieving and formatting GPS coordinates for further use, such as display or transmission in an IoT-based vehicle tracking system.

```
while (SerialGPS.available() > 0) {
   if (gps.encode(SerialGPS.read()))
   {
     if (gps.location.isValid())
     {
       latitude = gps.location.lat();
```

```
lat_str = String(latitude , 6);
longitude = gps.location.lng();
lng_str = String(longitude , 6);
```

- The OLED displays the latitude and longitude values with corresponding labels ("Lat:" and "Lng:").
- The GPS coordinates are also sent to the Blynk cloud on virtual pin V0 for remote monitoring. This ensures real-time visibility of the vehicle's location both locally (via the display) and remotely (via Blynk).

```
display.setTextAlignment(TEXT_ALIGN_LEFT);
display.setFont(ArialMT_Plain_16);
display.drawString(0, 23, "Lat:");
display.drawString(45, 23, lat_str);
display.drawString(0, 38, "Lng:");
display.drawString(45, 38, lng_str);
Blynk.virtualWrite(V0, 1, latitude, longitude, "Location");
```

This React-based web application is designed for a Vehicle Tracking System using IoT. The application fetches real-time GPS coordinates from Blynk IoT Cloud, where an ESP32 microcontroller sends latitude and longitude data using a GPS module.

Key Features of the Code:

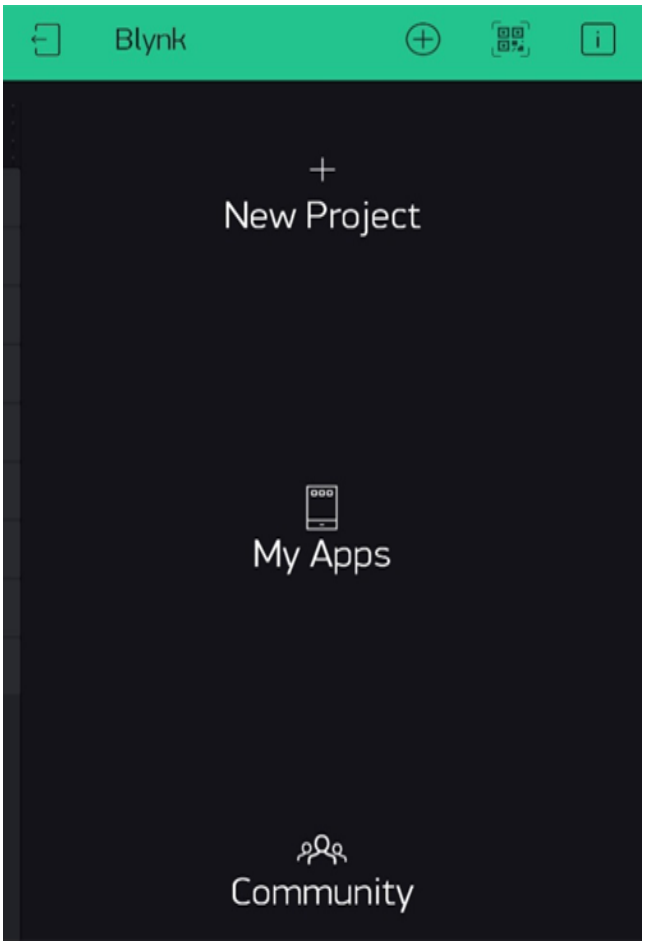
1. Fetches GPS Location Data
    - The fetchLocation function retrieves latitude (V1) and longitude (V2) from Blynk Cloud API.
    - The data is updated every 5 seconds using setInterval().
2. Displays Live Coordinates
    - The latitude and longitude value**s** are displayed on the web page dynamically.
3. Integrates with Google Maps
    - Uses an iframe to embed Google Ma**ps** and display the vehicle's real-time location.
4. Responsive UI with Tailwind CSS
    - The flexbox layout ensures the webpage is well-aligned and mobile-friendly.

```
import React, { useState, useEffect } from "react";


const VehicleTracking = () => {
  const [location, setLocation] = useState({ lat: 0, lng: 0 });
```

```
 useEffect(() => {
   const fetchLocation = async () => {
     try {
       const blynkAuth = "76aeba5832304e10917b4e1748c34039"; // Replace with your Blynk Auth
Token
       const latResponse = await fetch(`https://blynk.cloud/external/api/get?token=${blynkAuth}&V1`);
       const lngResponse = await fetch(`https://blynk.cloud/external/api/get?token=${blynkAuth}&V2`);

       const lat = await latResponse.text();
       const lng = await lngResponse.text();

       setLocation({ lat: parseFloat(lat), lng: parseFloat(lng) });
     } catch (error) {
       console.error("Error fetching location:", error);
     }
   };

   fetchLocation();
   const interval = setInterval(fetchLocation, 5000); // Auto-refresh every 5 sec
   return () => clearInterval(interval);
 }, []);

 return (
   <div className="flex flex-col items-center justify-center min-h-screen bg-gray-100 p-4">
     <h1 className="text-2xl font-bold mb-4">Blynk Vehicle Tracking</h1>
     <p className="text-lg mb-2">Latitude: {location.lat}</p>
     <p className="text-lg mb-4">Longitude: {location.lng}</p>
     <iframe
       title="Vehicle Location"
       width="600"
       height="450"
```

```
      style={{ border: 0 }}
      loading="lazy"
      allowFullScreen
      referrerPolicy="no-referrer-when-downgrade"
      src={`https://www.google.com/maps/embed/v1/place?key=AIzaSyDaGmWKa4JsXZ-HjGw7ISLn_3namBGewQe&q=${location.lat},${location.lng}`}
    ></iframe>
  </div>
 );
};

export default VehicleTracking;
```

6.4 Blynk Setup



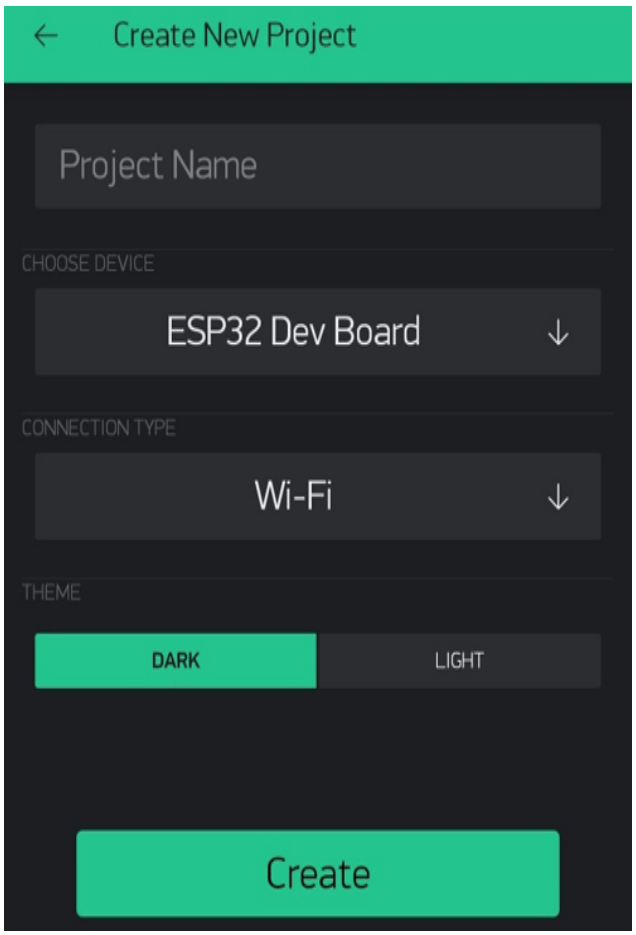
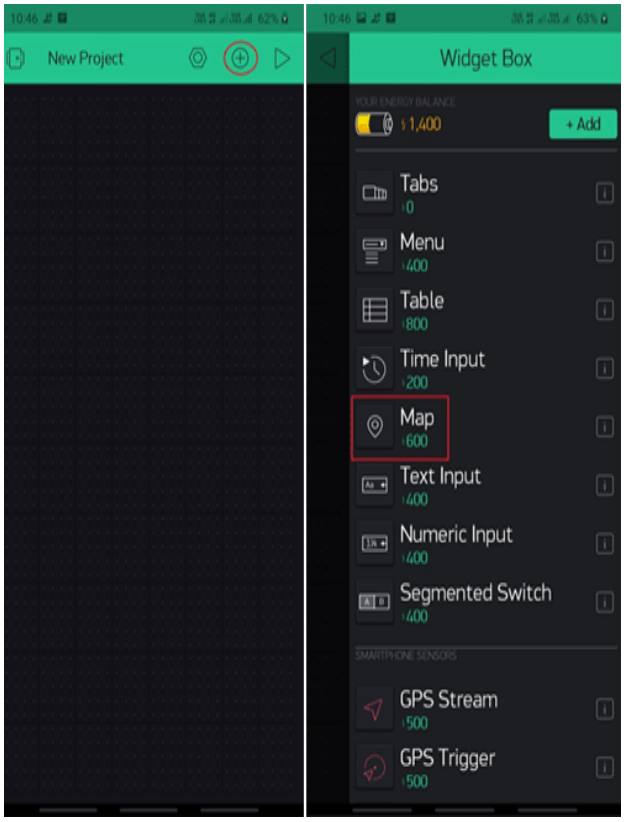


Fig. 11.1

Fig. 11.2



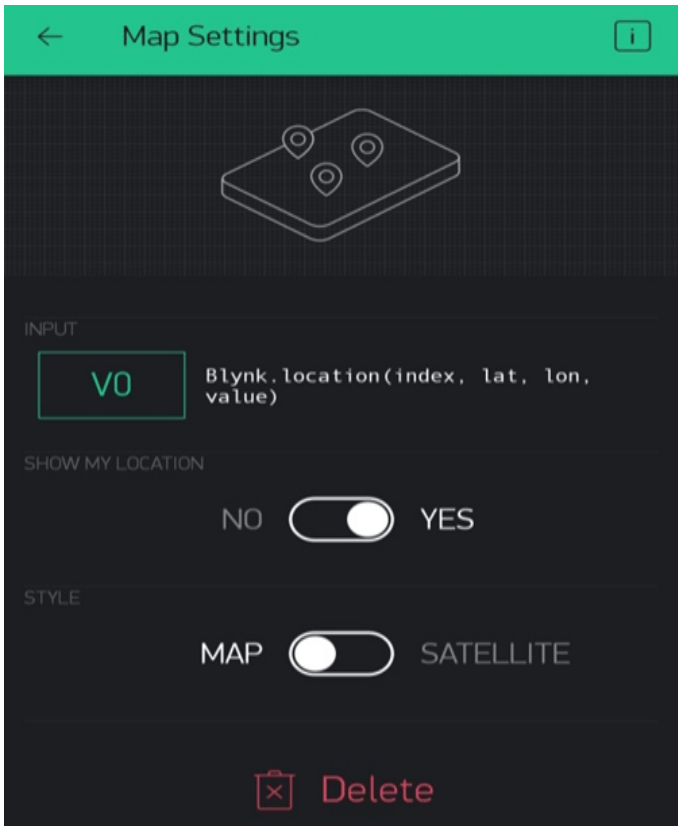


Fig. 11.3

Fig. 11.4

First Image(Fig. 11.1): Main Dashboard of the Blynk App

- The first screen is the Blynk app's main dashboard after logging in.
- Options available:
    1. New Project: Create a new IoT project.
    2. My Apps: View or manage existing projects or apps created.
    3. Community: Access the Blynk user community for help, discussions, or resources.

Second Image(Fig. 11.2): **Create New Project Interface**

- This screen appears when the "New Project" button is selected.
- Fields and options:
    1. Project Name: Input a custom name for the IoT project (e.g., "Vehicle Tracker").
    2. Choose Device: Select the hardware board being used (e.g., ESP32 Dev Board in the dropdown).
    3. Connection Type: Choose the connectivity type for the device (e.g., Wi-Fi, Bluetooth, etc.). Wi-Fi is selected here.
    4. Theme: Select the user interface theme:
        - Dark: A dark background theme (selected in the image).
        - Light: A light background theme.

5. Create Button: Finalize and create the project, which generates a Blynk Authentication Token for device setup.

Third Image(Fig. 11.3): Selecting Map Widget

- In this specific screen, you're looking at the "Widget Box." This is where you choose different types of widgets to add to your project. Widgets are like interactive elements that you can place on your Blynk app's interface.
- In the image, you can see a list of available widgets: Tabs, Menu, Table, Time Input, Map, Text Input, Numeric Input, Segmented Switch, GPS Stream, and GPS Trigger.
- To add a widget to your project, simply click on it in the list.

Fourth Image(fig. 11.4): Map Setting

INPUT:

- `V0`: This is a variable or pin used to define a location.
- `Blynk.location(...)`: This function is used to set the location's coordinates (latitude, longitude) and possibly other data.

- SHOW MY LOCATION**:** A toggle to show your current location on the map.
- STYLE**:** Choose between "MAP" (standard map) or "SATELLITE" view.

6.5 Outputs

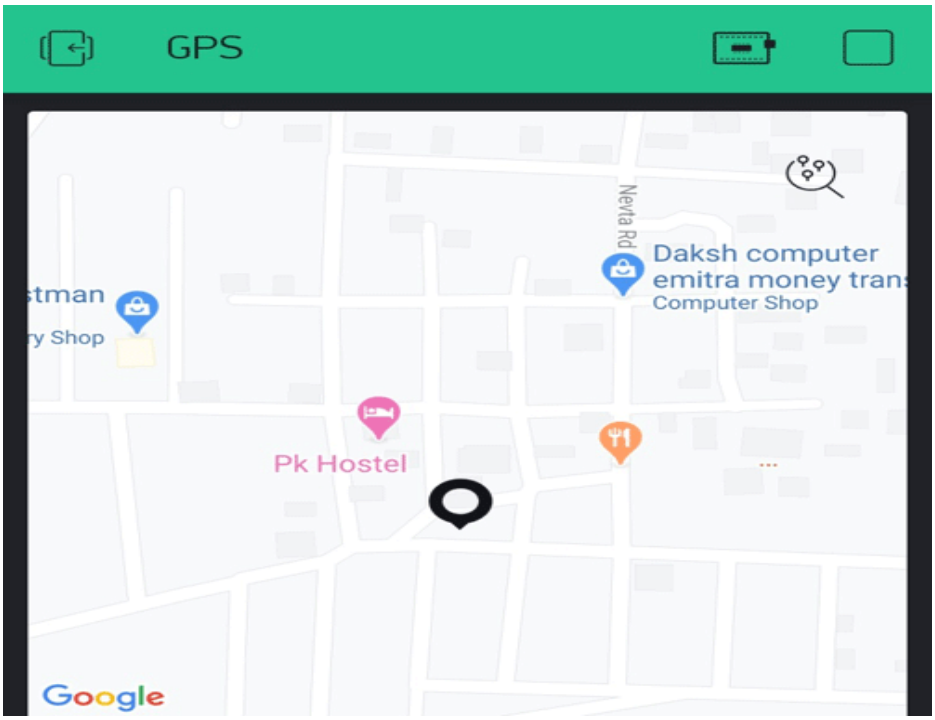


Fig. 12.1

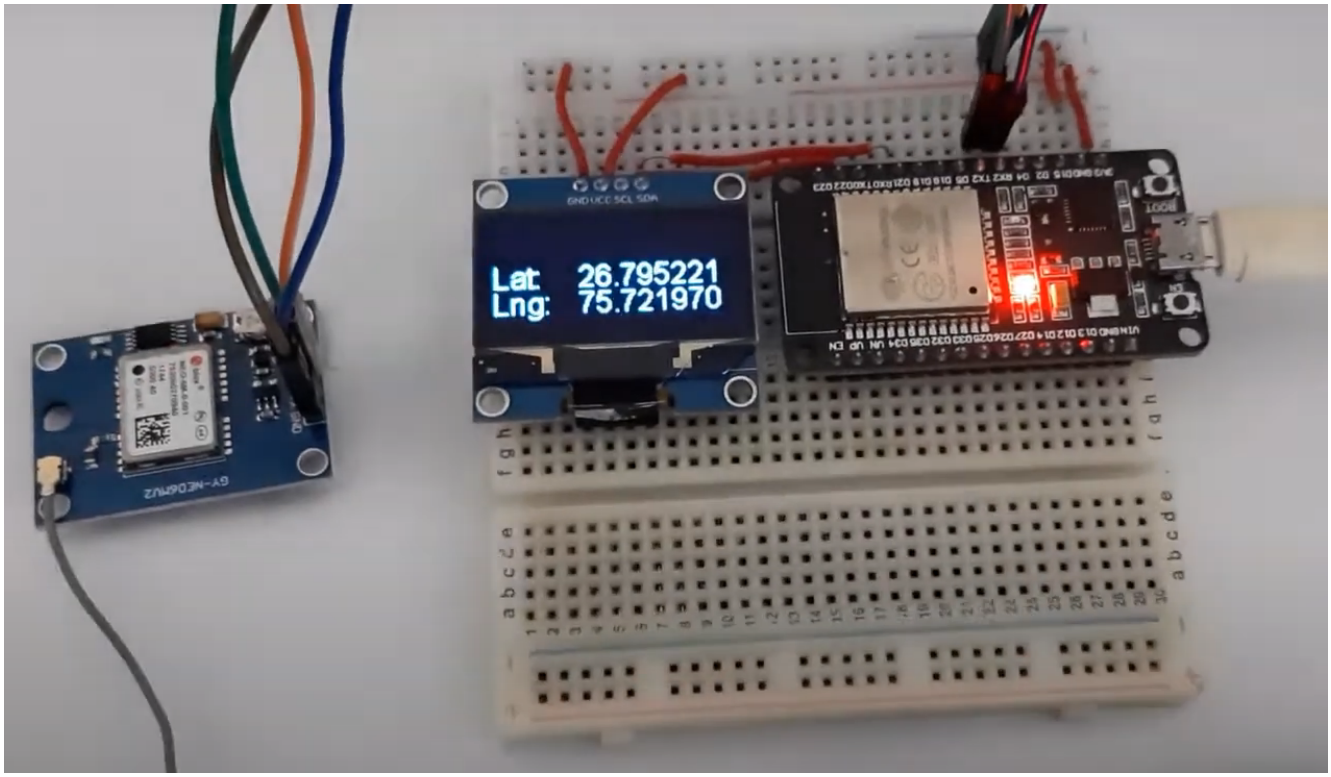


Fig. 12.2

CHAPTER 7: CONCLUSION AND FUTURE WORKS

7.1 Conclusions

The **Vehicle Tracking System Using IoT** has been successfully developed and implemented. This system integrates key components such as an ESP32 board, GPS and GSM modules, and the blynk software as cloud platform. It is user-friendly, cost-effective, and efficient, fulfilling all the objectives outlined at the project's inception.

The achievements of this project include:

- Developing a vehicle tracking system that is controlled via a smartphone and an embedded device.
- Creating a cost-effective yet highly efficient solution for vehicle tracking.
- Designing a user-friendly system that enhances safety, particularly for elder users who require additional support.

This project demonstrates the potential of IoT in addressing real-world problems while remaining accessible and affordable[2][4].

7.2 Future Recommendations

7.2.1 Enhancements

The future development of the IoT-based vehicle tracking system focuses on several potential Enhancements:

1. Integration with Vehicle Diagnostics: Adding advanced sensors to monitor critical vehicle parameters, such as fuel levels, engine health, and tire pressure. This feature would offer comprehensive vehicle management alongside tracking.
2. Use of LoRa Technology: Introducing LoRaWAN (Long Range Wide Area Network) communication technology to improve connectivity in remote and rural areas. This enhancement would reduce the dependency on GSM networks and enable more reliable tracking in areas with limited coverage.
3. Incorporation of Artificial Intelligence: Employing AI algorithms to predict vehicle maintenance needs, optimize routes, and analyze driver behavior. These capabilities can reduce operational costs and enhance safety[1][2][4].

7.2.2 Deployment at scale

1. **Scalability Testing**: Extending the system's deployment to larger fleets to evaluate its performance under diverse operational conditions. This includes testing for data handling capacity and network reliability.

2. **Industrial Collaborations**: Partnering with industries and transportation companies for large-scale implementations. These collaborations would provide practical feedback to refine the system further and ensure its suitability for various use cases[3].

References

1. Journal of Computer Networks and Communications:Proof of Concept of an IoT-Based Public Vehicle Tracking System, Using LoRa (Long Range) and Intelligent Transportation System (ITS) Services
https://onlinelibrary.wiley.com/doi/10.1155/2019/9198157

2. Arabian Journal of Science and Engineering: An IoT-Based School Bus and Vehicle Tracking System Using RFID Technology and Mobile Data Networks
https://link.springer.com/article/10.1007/s13369-020-05111-3

3. International Journal of Recent Technology and Engineering (IJRTE): IOT Based Vehicle Tracking and Monitoring System Using GPS and GSM
https://www.ijrte.org/wp-content/uploads/papers/v8i2S11/B12750982S1119.pdf

4. Research Gate: Vehicle Tracking System using Internet of Things
https://www.researchgate.net/publication/341192796_Vehicle_Tracking_System_using_Internet_of_Things