**SMART VEHICLE DIAGNOSTICS AND SMS-BASED LOCATION TRACKING SYSTEM**

A Project Report Submitted in Partial Fulfillment of Requirements to

**Centre for Development of Advanced Computing Hyderabad**

For the Award of the Degree of

**Post Graduate Diploma**

**In**

**Embedded Systems and Design**

Under the Esteemed Guidance of

**MR. Sarath V P**

**Submitted by**

Palakonda Venkata Sai Teja (250250330031)

Pinninti Lokesh (250250330032)

Poka Venkata Sai Jayanth (250250330033)

****

**FEBRUARY 2025 - AUGUST 2025**

## TABLE OF CONTENTS

**ABSTRACT**

1. **Introduction**

**1.1 Objectives**

**1.2 System Architecture**

**1.3 Working Principle**

1. **Implementation**
   1. **Components Used**
   2. **Component details**

**2.2.1. ESP 32 Micro Controller**

**2.2.2.MCP2515 CAN Controller**

**2.2.4. TJA1050 CAN Transceiver**

**2.2.5. JGB37-520 DC Motor (12V, 100 RPM)**

**2.2.6. L298N Motor Driver**

**3. Application Code**

**3.1 Vehicle diagnostics ESP 32 Transmitter Code**

**3.2 ESP 32 Receiver Code**

**4 . Features**

**5. Advantages**

**6. Applications**

**7. Conclusion & Future Scope**

1. **Abstract:**

The "Smart Vehicle Diagnostics and SMS-Based Location Tracking System" is an innovative solution designed to enhance vehicle safety and management. This project integrates real-time vehicle diagnostics, GPS-based location tracking, and remote control into a single embedded solution. The system uses an array of components including two ESP32 microcontrollers, a DC motor, and various sensors to monitor parameters like RPM, speed, fuel level, and temperature. This data is transmitted via CAN bus between the transmitter and receiver nodes. The system also features an accelerometer for accident (flip) detection, a GPS module for precise location tracking, and a GSM module for SMS-based communication. This allows users to request vehicle data, control the vehicle remotely, and receive automatic alerts in case of an accident. The project aims to provide a low-cost, reliable, and standalone system for comprehensive vehicle monitoring, making it suitable for applications such as fleet management and school bus tracking.

1. **Introduction:**

The rapid advancement in automotive electronics and IoT integration has created opportunities for developing intelligent vehicle monitoring and control systems. Traditional vehicle diagnostics and GPS tracking solutions often exist as separate modules, increasing system complexity and cost. This project integrates real-time vehicle diagnostics, GPS-based location tracking, and remote control into a single embedded solution. The system utilizes a combination of CAN bus communication, GPS modules, GSM modules, and sensor data processing to provide the user with live vehicle parameters, accident detection alerts, and location details via SMS.

1. **Objectives:**

The primary objectives of this project are: To design a low-cost, reliable embedded system for vehicle diagnostics and tracking. To transmit real-time parameters like RPM, speed, fuel level, and temperature using CAN bus. To implement SMS-based communication for remote control and data requests. To integrate GPS for precise location tracking. To include accident detection (flip detection) using an accelerometer.

1. **System Architecture:**

The system is designed with a two-node architecture: a transmitter node and a receiver node.

**Transmitter Node:**

Connected to the motor, encoder, temperature sensor, and fuel sensor. Reads RPM, speed, temperature, and fuel level. Sends this data over the CAN bus to the receiver node. Can be controlled remotely via motor ON/OFF commands from the receiver. **Receiver Node:**

Equipped with GPS, GSM, accelerometer, and CAN module. Receives CAN data from the transmitter. Monitors the accelerometer for flip detection. Responds to SMS commands (hi, start, stop) by sending GPS location or vehicle diagnostics. Sends accident alerts automatically to predefined numbers.

1. **Working Principle:**

The system operates based on the following principles:

**Vehicle Diagnostics (Transmitter Side):**

The encoder provides pulse counts, which are converted into RPM and speed. The temperature sensor outputs a voltage proportional to the temperature (for an LM35, this is 10mV/°C). The fuel sensor outputs a varying voltage based on the simulated tank level. All these parameters are packed into an 8-byte CAN frame and sent to the receiver.

**Data Reception (Receiver Side):**

The CAN module reads the incoming data frame and decodes the RPM, speed, temperature, and fuel level. This data is then stored for SMS replies or for local display.

**GPS Location Tracking:**

The GPS module constantly updates the latitude and longitude. This location information is formatted as a Google Maps link for transmission via SMS.

**SMS Communication:**

The SIM900A module receives SMS commands from the user. The supported commands include:

"hi" – returns the GPS location (in GPS mode) or vehicle diagnostics (in CAN mode). "start" – sends a CAN command to start the motor.

"stop" – sends a CAN command to stop the motor.

**Accident Detection:**

The accelerometer reads the X, Y, and Z axis values every 5 seconds. If the Z-axis value drops below a certain threshold or the X-axis value exceeds a set limit, the system assumes a flip accident has occurred. It then sends an alert SMS with the GPS location to predefined contacts.

**Mode Switching:**

A push button is used to toggle between GPS mode (for location data) and CAN mode (for vehicle diagnostics).

# Components Used in Project

|  |  |
| --- | --- |
| Component | Purpose/Function |
| ESP32 Microcontroller | Main controller for transmitter & receiver nodes, handles sensors, CAN, GPS, GSM. |
| MCP2515 CAN Controller | Interface between ESP32 (SPI) and CAN bus protocol. |
| TJA1050 CAN Transceiver | Converts MCP2515 signals to differential CAN\_H & CAN\_L levels. |
| JGB37-520 DC Motor (12V, 100 RPM) | Represents vehicle wheel/motor for RPM and speed calculation. |
| L298N Motor Driver | Drives the DC motor, enables forward/reverse and speed control. |
| Rotary Encoder | Provides pulses for motor RPM and speed measurement. |
| NTC Temperature Sensor | Monitors engine/ambient temperature for diagnostics. |
| Fuel Level Sensor (Potentiometer) | Simulates fuel tank level for diagnostics. |
| NEO-6M GPS Module | Provides latitude and longitude for vehicle tracking. |
| SIM900A GSM Module | Sends/receives SMS for alerts, location, and control commands. |
| Accelerometer (Analog) | Detects car flip/tilt for accident alert system. |
| Push Button | Switches between GPS mode and Diagnostics mode. |
| CAN Bus Cables & Connectors | Used for CAN communication wiring between nodes. |
| Power Supply (12V + 5V Regulator) | Provides required power to ESP32, sensors, motor, and modules. |

**ESP 32 Microcontroller :**

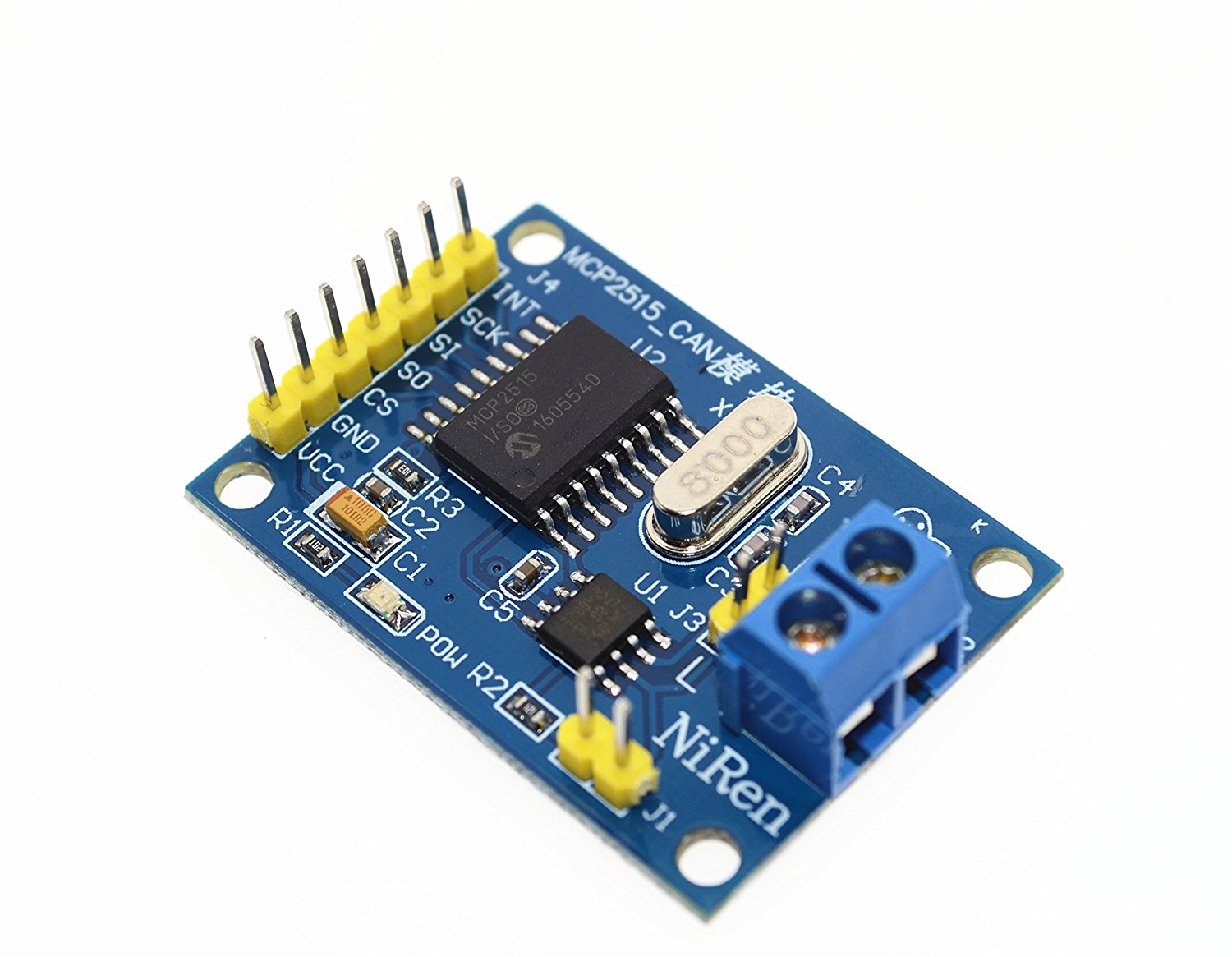
****

The ESP32 is a low-cost, low-power, and highly versatile microcontroller developed by Espressif Systems. It is built on a dual-core Tensilica Xtensa LX6 processor, with clock speeds up to 240 MHz, making it suitable for both real-time embedded applications and higher-level IoT tasks.

Key specifications relevant to our project include:

* **Processing Power:** Dual-core processor with integrated FPU, enabling efficient handling of sensor data, CAN communication, and GSM/GPS coordination in real time.
* **Memory:** 520 KB SRAM and up to 16 MB external flash support, sufficient for complex logic, data buffering, and protocol stacks.
* **Connectivity:** Native **Wi-Fi** and **Bluetooth Low Energy (BLE)** interfaces, although in this project GSM was chosen for SMS-based control due to reliability in outdoor vehicle environments.
* **GPIO Flexibility:** Over 30 GPIO pins configurable for ADC, DAC, PWM, I²C, SPI, UART, and CAN, allowing the ESP32 to interface seamlessly with multiple peripherals.
* **Analog Inputs:** 12-bit ADC (18 channels), crucial for reading signals from temperature sensors, accelerometers, and fuel-level sensors.
* **Timers and Interrupts:** Hardware timers and external interrupt support, used here for precise encoder pulse counting to calculate motor RPM.
* **Low Power Modes:** Multiple deep-sleep modes make it suitable for automotive and IoT systems where power saving is critical.

**MCP2515 CAN Controller :**



The MCP2515 is a stand-alone Controller Area Network (CAN) controller manufactured by Microchip Technology. It implements the CAN protocol (version 2.0B), which supports both the standard 11-bit identifier and the extended 29-bit identifier. Since the ESP32 does not natively have a CAN controller (it has a TWAI interface, but we used MCP2515 for flexibility), the MCP2515 acts as the bridge between ESP32 (via SPI) and the CAN bus (via CAN transceiver like TJA1050).

#### Key Features

* **Protocol Support:** Implements CAN 2.0A (11-bit ID) and CAN 2.0B (29-bit ID).
* **SPI Interface:** Uses SPI protocol (up to 10 MHz) for communication with the ESP32.
* **Transmit/Receive Buffers:**
* 3 Transmit buffers (TXB0, TXB1, TXB2).
* 2 Receive buffers (RXB0, RXB1).
* **Filters & Masks:**
* 6 acceptance filters and 2 masks for selective message filtering (reduces CPU load).
* **Interrupt Support:** Interrupt pin alerts the microcontroller when a message is received, transmitted, or if an error occurs.
* **Error Handling:** Supports error counters, bus-off state, and automatic retransmission.
* **Baud Rate Flexibility:** Works up to 1 Mbps depending on the CAN transceiver.
* **Low Power Mode:** Can enter sleep mode for automotive/embedded applications.

**TJA1050 CAN Transceiver :**

The TJA1050 is a high-speed CAN transceiver manufactured by NXP Semiconductors. While the MCP2515 handles the data-link layer (frame formatting, arbitration, error checking), the TJA1050 handles the physical layer of the CAN protocol. It is responsible for converting the logic-level CAN signals from MCP2515 into the differential signals required on the CAN bus lines (CAN\_H and CAN\_L).

### Key Features

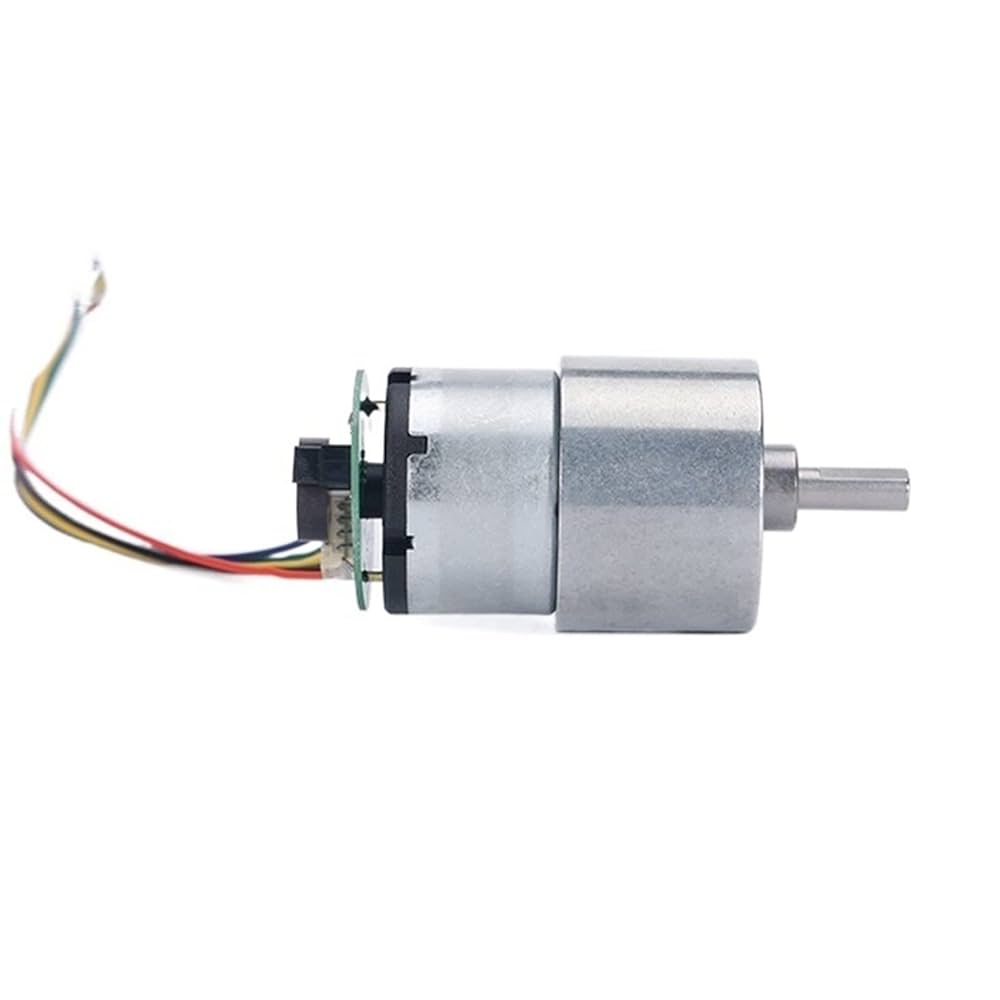
* **High-Speed Operation:** Supports baud rates up to **1 Mbps**, compliant with ISO 11898 standard.
* **Differential Output:** Converts single-ended logic signals (TXD, RXD) to **differential voltages** across CAN\_H and CAN\_L.
* **Voltage Levels:**

**Recessive state:** CAN\_H ≈ 2.5 V, CAN\_L ≈ 2.5 V (no dominant bit).

**Dominant state:** CAN\_H ≈ 3.5 V, CAN\_L ≈ 1.5 V (differential = ~2 V).

* **Low Power Mode:** Supports standby mode for reduced current consumption in automotive systems.
* **Short-Circuit Protection:** Protects against shorts of CAN\_H or CAN\_L to ground or Vcc.
* **Thermal Protection:** Safeguards the transceiver from overheating.
* **Electromagnetic Compatibility (EMC):** Designed to minimize electromagnetic interference (important in automotive environments).

**JGB37-520 DC Motor (12V, 100 RPM) :**

****

The **JGB37-520** is a **DC geared motor** commonly used in robotics, automation, and embedded systems. It combines a **12V brushed DC motor** with a **metal gearbox** to reduce speed and increase torque, making it suitable for applications requiring **controlled rotational speed and higher load capacity**.

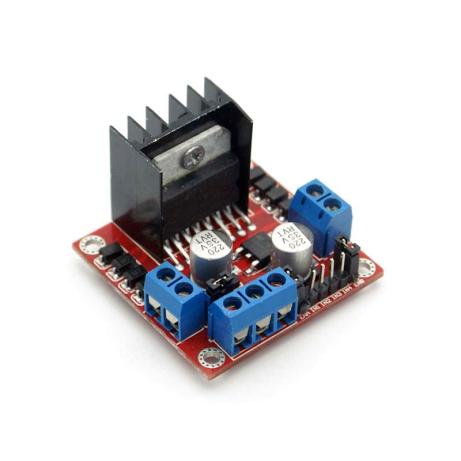
### Key Specifications

* **Operating Voltage:** 12V DC
* **Rated Speed:** ~100 RPM (after gearbox reduction)
* **Gearbox Type:** Metal, high durability
* **Torque:** High output torque due to gear reduction (varies depending on load, approx. 3–10 kg·cm)
* **Shaft Diameter:** ~6 mm, D-type shaft for secure coupling
* **Motor Type:** Brushed DC motor (easy to control with PWM and motor drivers)
* **Gear Ratio:** Around 1:50 (depends on exact model variant)

### Working Principle

* The motor runs on **DC power (12V)**.
* The **brushed DC mechanism** converts electrical energy into mechanical rotational energy using commutators and brushes.
* The **metal gearbox** reduces the high speed of the DC motor and increases torque.
* The final output is a **low-speed, high-torque rotation** (~100 RPM), making it suitable for driving wheels or loads that require strength rather than high speed.

**L298N Motor Driver**



The **L298N** is a popular **dual H-Bridge motor driver IC**, widely used to control the speed and direction of DC motors and stepper motors. It allows low-power microcontrollers (like ESP32, Arduino, etc.) to control higher-power motors by acting as an **interface between logic-level control signals and the high-current motor load**.

### Key Specifications

* **Operating Voltage (Vcc):** 5V – 35V (logic supply usually 5V, motor supply up to 35V)
* **Output Current:** Up to 2A per channel (continuous), peak 3A
* **Number of Channels:** 2 (can drive two DC motors independently)
* **Logic Input Voltage:** 3.3V / 5V compatible (ESP32 works fine)
* **Control Type:** Dual H-Bridge → Forward, Reverse, Stop, Brake
* **PWM Support:** Allows speed control using PWM signals from a microcontroller
* **Built-in Protection:** Includes diodes for back-EMF protection (important for inductive loads like motors)

Working Principle

The **H-Bridge** is the core of the L298N driver. An H-Bridge is a circuit that allows current to flow in **both directions** through a motor, enabling it to spin **forward or backward**.

1. **Inputs (IN1, IN2, IN3, IN4):** Decide the motor rotation direction.

* Example: IN1 = HIGH, IN2 = LOW → Motor rotates forward.
* IN1 = LOW, IN2 = HIGH → Motor rotates backward.

1. **Enable Pins (ENA, ENB):** Act as ON/OFF switches for the motors. They are usually controlled with **PWM signals** for speed control.
2. **Outputs (OUT1–OUT4):** Connect to the motor terminals.
3. **Power Pins:**

* **Vcc:** Motor supply (up to 35V).
* **5V (Logic):** Logic power supply (for ESP32 control).
* **GND:** Common ground with microcontroller.

When a logic HIGH/LOW is applied to the input pins, the H-Bridge transistors switch and control current direction in the motor. By applying **PWM to ENA/ENB**, the motor speed can be varied.

**Rotary Encoder**

A **rotary encoder** is an electro-mechanical sensor used to convert the angular position or motion of a shaft into electrical signals. In this project, it is used with the **JGB37-520 DC Motor** to measure the **speed (RPM)** and indirectly calculate the **vehicle speed**.

### Key Specifications

* **Type:** Incremental Rotary Encoder (used with DC motor shaft)
* **Resolution (PPR):** ~550 pulses per revolution (depending on model)
* **Output Signal:** Quadrature (two square-wave signals, A and B, phase-shifted by 90°)
* **Voltage Supply:** 5V typical
* **Output Type:** Digital (TTL/CMOS compatible, works with ESP32 GPIOs)
* **Shaft Mounting:** Directly coupled to motor shaft

### Working Principle

* An **incremental encoder** works by generating **a series of digital pulses** as the motor shaft rotates.
* Typically, it produces **two outputs (A and B channels)** that are 90° out of phase (quadrature signals).
* By counting the pulses, the **microcontroller (ESP32)** can determine how many times the shaft has rotated.
* By measuring the **time interval between pulses**, the **speed (RPM)** can be calculated.
* If both A and B channels are read, the **direction of rotation** can also be determined.
* In our project, we mainly use it for **RPM calculation**.

**NEO-6M GPS Module**

****

The **NEO-6M GPS module** is a high-performance positioning receiver developed by u-blox. It provides accurate location and timing information by receiving signals from multiple satellites. In our project, it is used on the **Receiver ESP32 board** to provide **real-time vehicle location tracking**, which can then be sent to the user via SMS.

## Key Specifications

* **Chipset:** u-blox NEO-6M
* **Positioning Accuracy:**
  + 2.5 m (CEP, without SBAS)
  + < 2.0 m (with SBAS: WAAS, EGNOS, MSAS)
* **Update Rate:** 1 Hz (default), up to 5 Hz
* **Receiver Sensitivity:**
  + Tracking: −161 dBm
  + Cold start: −147 dBm
* **Cold Start Time:** ~27 seconds (typical)
* **Hot Start Time:** ~1 second
* **Operating Voltage:** 2.7 V – 3.6 V (commonly powered via onboard regulator at 3.3 V or 5 V)
* **Communication Interface:** UART (default), also supports SPI/I²C (advanced configs)
* **Antenna:** External ceramic patch antenna or active antenna (via SMA connector)

## Working Principle

* The NEO-6M continuously listens to signals from **multiple GPS satellites** orbiting Earth.
* By calculating the **time delay** between signals from at least four satellites, it performs **trilateration** to determine:
  + Latitude
  + Longitude
  + Altitude
  + UTC Time
* Speed and course (if moving)
* The module outputs data in **NMEA format (ASCII sentences)** such as $GPGGA, $GPRMC, etc.
* The **ESP32 (Receiver)** reads this serial data (via UART) and processes it using the **TinyGPS++ library**, which extracts location, time, and speed.
* The processed GPS coordinates are converted into **Google Maps links** for user convenience.

**SIM900A GSM Module**

****

The **SIM900A GSM module** is a compact and reliable **dual-band GSM/GPRS modem** designed by SIMCom. It allows microcontrollers and embedded systems to communicate over the **GSM cellular network** for services such as **SMS, voice calls, and GPRS-based data transfer**. In our project, it plays a central role in enabling **SMS-based communication between the vehicle and the user**.

## Key Specifications

* **Frequency Bands:** Dual-band GSM 900 MHz and 1800 MHz (commonly used in India and Asia).
* **Supply Voltage:** 3.4 V – 4.5 V (typical: 4.0 V).
* **Communication Interface:** UART (AT command set).
* **Baud Rate:** Default 9600 bps (configurable up to 115200 bps).
* **Features Supported:**
  + SMS (Text / PDU mode)
  + Voice calls
  + GPRS data (TCP/IP stack)
  + USSD
* **Power Consumption:**
  + Idle mode: ~1.5 mA
  + Active (voice/SMS): ~250 mA
  + Peak: 2 A (during transmission bursts).

## Working Principle

* The SIM900A contains a GSM modem and requires a **SIM card** with active cellular service.
* It connects to the **cellular network** by registering with the nearest GSM tower.
* It communicates with the **ESP32** over **UART** using a standard set of **AT commands**:
  + AT → Module check
  + AT+CMGF=1 → Set SMS mode (Text mode)
  + AT+CMGS="number" → Send SMS to a phone number
  + AT+CNMI=2,2,0,0,0 → Configure SMS reception

In our project:

* On receiving commands like "hi", "start", "stop" via SMS, the ESP32 processes them and responds appropriately.
* It also sends **automatic SMS alerts** when the **accelerometer detects a car flip**, attaching **GPS coordinates** from the NEO-6M module.

**CODE:**

**Transmitter code:**

#include <SPI.h>

#include <mcp\_can.h>

// Motor control pins

#define ENA 13

#define IN1 12

#define IN2 14

// Encoder pin

#define ENCODER\_A 25

// Analog input pins

#define TEMP\_PIN 34

#define FUEL\_PIN 26

// CAN setup

#define CAN\_CS 5

MCP\_CAN CAN(CAN\_CS);

const float wheel\_diameter\_cm = 30.0;

const float wheel\_circumference\_m = 3.1416 \* (wheel\_diameter\_cm / 100.0);

const int pulses\_per\_revolution = 550;

volatile int encoderCount = 0;

unsigned long prevTime = 0;

bool motorRunning = true;

void IRAM\_ATTR readEncoderA() {

encoderCount++;

}

void setup() {

Serial.begin(115200);

pinMode(ENA, OUTPUT);

pinMode(IN1, OUTPUT);

pinMode(IN2, OUTPUT);

pinMode(ENCODER\_A, INPUT\_PULLUP);

attachInterrupt(digitalPinToInterrupt(ENCODER\_A), readEncoderA, RISING);

digitalWrite(IN1, HIGH);

digitalWrite(IN2, LOW);

analogWrite(ENA, 255); // Start motor

while (CAN\_OK != CAN.begin(MCP\_ANY, CAN\_500KBPS, MCP\_8MHZ)) {

Serial.println("CAN Init Failed");

delay(500);

}

Serial.println("CAN Init Success");

CAN.setMode(MCP\_NORMAL);

}

void loop() {

// Listen for control commands

if (CAN\_MSGAVAIL == CAN.checkReceive()) {

long unsigned int rxId;

byte len = 0;

byte buf[8];

CAN.readMsgBuf(&rxId, &len, buf);

if (rxId == 0x101 && len >= 1) {

if (buf[0] == 0x00) {

Serial.println("Received STOP command");

analogWrite(ENA, 0);

motorRunning = false;

} else if (buf[0] == 0x01) {

Serial.println("Received START command");

analogWrite(ENA, 255);

motorRunning = true;

}

}

}

if (millis() - prevTime >= 1000) {

prevTime = millis();

float rpm = 0, speed\_kmph = 0;

if (motorRunning) {

rpm = (encoderCount \* 60.0) / pulses\_per\_revolution;

float speed\_mps = (rpm \* wheel\_circumference\_m) / 60.0;

speed\_kmph = speed\_mps \* 3.6;

}

int rpm\_encoded = (int)(rpm \* 10);

int speed\_encoded = (int)(speed\_kmph \* 10);

int temp\_adc = analogRead(TEMP\_PIN);

float voltage = (temp\_adc / 4095.0) \* 3.3;

float temp\_c = voltage \* 100.0;

int temp\_encoded = (int)(temp\_c \* 10);

int fuel\_adc = analogRead(FUEL\_PIN);

int fuel\_percent = map(fuel\_adc, 0, 4095, 0, 100);

Serial.print("RPM: "); Serial.print(rpm, 1);

Serial.print("\tSpeed: "); Serial.print(speed\_kmph, 1); Serial.print(" km/h");

Serial.print("\tTemp: "); Serial.print(temp\_c, 1); Serial.print(" °C");

Serial.print("\tFuel: "); Serial.print(fuel\_percent); Serial.println(" %");

// CAN data packet

byte data[8];

data[0] = (rpm\_encoded >> 8) & 0xFF;

data[1] = rpm\_encoded & 0xFF;

data[2] = (speed\_encoded >> 8) & 0xFF;

data[3] = speed\_encoded & 0xFF;

data[4] = (temp\_encoded >> 8) & 0xFF;

data[5] = temp\_encoded & 0xFF;

data[6] = fuel\_percent;

data[7] = 0; // Reserved or unused

CAN.sendMsgBuf(0x100, 0, 8, data);

encoderCount = 0;

}

}

**Receiver Code:**

#include <Arduino.h>

#include <TinyGPS++.h>

#include <SPI.h>

#include <mcp\_can.h>

// --------------- Pin Definitions ---------------

#define GPS\_RX 4

#define GPS\_TX 2

#define SIM900\_RX 16

#define SIM900\_TX 17

#define CAN\_CS 5

#define X\_PIN 34

#define Y\_PIN 35

#define Z\_PIN 32

#define BUTTON\_PIN 15 // Button to switch modes

// --------------- GPS & GSM Setup ---------------

TinyGPSPlus gps;

HardwareSerial gpsSerial(1);

HardwareSerial sim900(2);

// --------------- CAN Bus Setup ---------------

MCP\_CAN CAN(CAN\_CS);

float rpm = 0, speed\_kmph = 0, temp\_c = 0;

int fuel\_percent = 0;

// --------------- SMS Handling ---------------

String incomingSMS = "";

String senderNumber = "";

bool awaitingMessage = false;

// --------------- Accelerometer Setup ---------------

bool messageSent = false;

static unsigned long lastAlertTime = 0;

const unsigned long alertCooldown = 10000; // 10s cooldown between flip alerts

// Phone numbers for car flip alert

const char\* phoneNumbers[] = {

"+919492099635", "+918978693756"

};

const int numRecipients = sizeof(phoneNumbers) / sizeof(phoneNumbers[0]);

// --------------- AT Command Helper ---------------

void sendATCommand(const char \*cmd, unsigned long delayMs = 1000) {

sim900.println(cmd);

delay(delayMs);

while (sim900.available()) Serial.write(sim900.read());

Serial.println();

}

void sendSMS(const char\* phoneNumber, const char\* message) {

sendATCommand("AT+CMGF=1", 1000);

sim900.print("AT+CMGS=\"");

sim900.print(phoneNumber);

sim900.println("\"");

delay(1000);

sim900.print(message);

sim900.write(26); // Ctrl+Z

delay(5000);

while (sim900.available()) Serial.write(sim900.read());

Serial.println("SMS sent.");

}

void configureSIM900ForSMS() {

sendATCommand("AT");

sendATCommand("AT+CMGF=1");

sendATCommand("AT+CNMI=2,2,0,0,0");

}

// --------------- Motor Control ---------------

void sendMotorCommand(bool start) {

byte cmd[1] = { start ? 0x01 : 0x00 };

CAN.sendMsgBuf(0x101, 0, 1, cmd);

Serial.println(start ? "Motor START command sent" : "Motor STOP command sent");

}

// --------------- GPS Location Helper ---------------

String getGPSLocation() {

if (gps.location.isValid()) {

float lat = gps.location.lat();

float lng = gps.location.lng();

return "Lat: " + String(lat, 6) + ", Lng: " + String(lng, 6) +

"\nMap: https://maps.google.com/?q=" + String(lat, 6) + "," + String(lng, 6);

} else {

return "GPS fix not available.";

}

}

// --------------- Extract Sender Number ---------------

String extractSenderNumber(const String &sms) {

int idx1 = sms.indexOf('"');

int idx2 = sms.indexOf('"', idx1 + 1);

if (idx1 >= 0 && idx2 > idx1) {

return sms.substring(idx1 + 1, idx2);

}

return "";

}

// --------------- Setup ---------------

void setup() {

Serial.begin(115200);

analogReadResolution(12);

pinMode(BUTTON\_PIN, INPUT\_PULLUP);

gpsSerial.begin(9600, SERIAL\_8N1, GPS\_RX, GPS\_TX);

sim900.begin(9600, SERIAL\_8N1, SIM900\_RX, SIM900\_TX);

delay(3000);

configureSIM900ForSMS();

while (CAN\_OK != CAN.begin(MCP\_ANY, CAN\_500KBPS, MCP\_8MHZ)) {

Serial.println("CAN Init Failed");

delay(500);

}

CAN.setMode(MCP\_NORMAL);

Serial.println("CAN Init Success");

Serial.println("Send 'hi', 'start', or 'stop' via SMS to control device.");

}

// --------------- Main Loop ---------------

void loop() {

bool buttonPressed = (digitalRead(BUTTON\_PIN) == LOW);

// GPS feed if in GPS mode

if (!buttonPressed) {

while (gpsSerial.available()) gps.encode(gpsSerial.read());

}

// Accelerometer Flip Detection only in GPS mode

if (!buttonPressed) {

static unsigned long lastAccelCheck = 0;

if (millis() - lastAccelCheck > 5000) {

lastAccelCheck = millis();

int x = analogRead(X\_PIN);

int y = analogRead(Y\_PIN);

int z = analogRead(Z\_PIN);

bool flipped = (z < 1600 || x > 2100);

if (flipped && (millis() - lastAlertTime > alertCooldown)) {

String alertMsg = "ALERT: Car Flipped!\nX:" + String(x) +

" Y:" + String(y) + " Z:" + String(z) +

"\n" + getGPSLocation();

for (int i = 0; i < numRecipients; i++) {

sendSMS(phoneNumbers[i], alertMsg.c\_str());

delay(5000);

}

lastAlertTime = millis();

}

}

}

// Check SIM900 for incoming SMS

while (sim900.available()) {

String line = sim900.readStringUntil('\n');

line.trim();

if (line.length() == 0) continue;

Serial.println("SIM900 >> " + line);

if (line.startsWith("+CMT:")) {

senderNumber = extractSenderNumber(line);

awaitingMessage = true;

}

else if (awaitingMessage) {

awaitingMessage = false;

incomingSMS = line;

incomingSMS.toLowerCase();

incomingSMS.trim();

String reply;

if (incomingSMS == "hi") {

if (!buttonPressed) {

reply = getGPSLocation(); // Only GPS in GPS mode

} else {

reply = "RPM: " + String(rpm, 1) +

"\nSpeed: " + String(speed\_kmph, 1) + " km/h" +

"\nTemp: " + String(temp\_c, 1) + " °C" +

"\nFuel: " + String(fuel\_percent) + " %";

}

} else if (incomingSMS == "start") {

sendMotorCommand(true);

reply = "Motor STARTED.";

} else if (incomingSMS == "stop") {

sendMotorCommand(false);

reply = "Motor STOPPED.";

} else {

reply = "Commands:\nhi - GPS or CAN data\nstart - Motor ON\nstop - Motor OFF";

}

sendSMS(senderNumber.c\_str(), reply.c\_str());

}

}

// Manual serial motor test

if (Serial.available()) {

char c = Serial.read();

if (c == 'r') sendMotorCommand(true);

else if (c == 's') sendMotorCommand(false);

}

// CAN Data Receive

if (CAN\_MSGAVAIL == CAN.checkReceive()) {

long unsigned int rxId;

byte len = 0;

byte buf[8];

CAN.readMsgBuf(&rxId, &len, buf);

if (rxId == 0x100 && len == 8) {

int rpm\_encoded = (buf[0] << 8) | buf[1];

int speed\_encoded = (buf[2] << 8) | buf[3];

int temp\_encoded = (buf[4] << 8) | buf[5];

fuel\_percent = buf[6];

rpm = rpm\_encoded / 10.0;

speed\_kmph = speed\_encoded / 10.0;

temp\_c = temp\_encoded / 10.0;

Serial.print("RPM: "); Serial.print(rpm);

Serial.print(" Speed: "); Serial.print(speed\_kmph); Serial.print(" km/h");

Serial.print(" Temp: "); Serial.print(temp\_c); Serial.print(" °C");

Serial.print(" Fuel: "); Serial.print(fuel\_percent); Serial.println(" %");

}

}

}

**Features:**

The system boasts the following features:

1. Dual-mode operation (GPS mode / CAN data mode).
2. Two-way communication: the user can send commands, and the system can send alerts.
3. Accident alerts with location details.
4. Low-latency CAN bus communication between the transmitter and receiver. Standalone operation without the need for an internet connection.

**Advantages:**

1. The advantages of this system include:
2. Low cost due to the use of readily available modules.
3. Reliable due to the wired CAN communication between the nodes.
4. SMS functionality works in areas without internet access.
5. Real-time accident alerts improve safety.

**Applications:**

This system has a wide range of applications, including:

1. Fleet vehicle monitoring.
2. School bus tracking and safety systems.
3. Remote agricultural vehicle monitoring.
4. Logistics vehicle diagnostics and tracking.

**Conclusion and Future Scope:**

The "Smart Vehicle Diagnostics and SMS-Based Location Tracking System" successfully integrates vehicle diagnostics, location tracking, and remote control into a single, efficient system. By leveraging the capabilities of the ESP32, CAN bus, GPS, and GSM modules, the project provides a reliable and low-cost solution for real-time vehicle monitoring. The system's ability to operate without an internet connection and provide real-time accident alerts significantly enhances vehicle safety and management.

**Future enhancements could include:**

1. **Integration with a mobile application** for a more user-friendly interface.
2. **Cloud data logging** for historical data analysis and predictive maintenance.
3. **Expansion with more sensors** to monitor other vehicle parameters like tire pressure and engine fault codes.
4. **Implementation of more advanced accident detection algorithms** to reduce false positives.