AUTONOMOUS VEHICLE TRACKING AND MONITORING USING IoT

Project Documentation

An IoT-based Solution for Real-Time Vehicle Tracking and Environmental Monitoring in Agricultural Applications

Submitted By:

SAICHARITHA DATHRIKA

Date: May 21, 2021

Table of Contents

1. INTRODUCTION

In an agricultural environment, heavy vehicles like harvesters and tractors are used to perform farming-related activities. Using IoT (Internet of Things), diagnostic information of the vehicle can be shared to a cloud platform, and the GPS location of the vehicle is also transmitted to the server.

These received data will be analyzed and are helpful for the owner of the vehicles to make sure necessary actions are taken to minimize downtime. This project implements a real-time vehicle tracking and monitoring system using IoT technologies.

The system uses NodeMCU (ESP8266) as the main microcontroller, which connects to the internet via Wi-Fi and sends GPS coordinates and environmental data (temperature and humidity) to the ThingSpeak cloud platform for visualization and analysis.

2. COMPONENTS USED

HARDWARE COMPONENTS	SOFTWARE COMPONENTS
• NodeMCU (ESP8266) • L293D Motor Driver • BO Motors (4 units) • Power Supply (Li-ion Batteries) • DHT11 Temperature & Humidity Sensor • NEO-6M GPS Module • Jumper Wires	• Arduino IDE • ThingSpeak IoT Platform • Blynk Mobile App

3. HARDWARE COMPONENTS

3.1 NodeMCU (ESP8266)

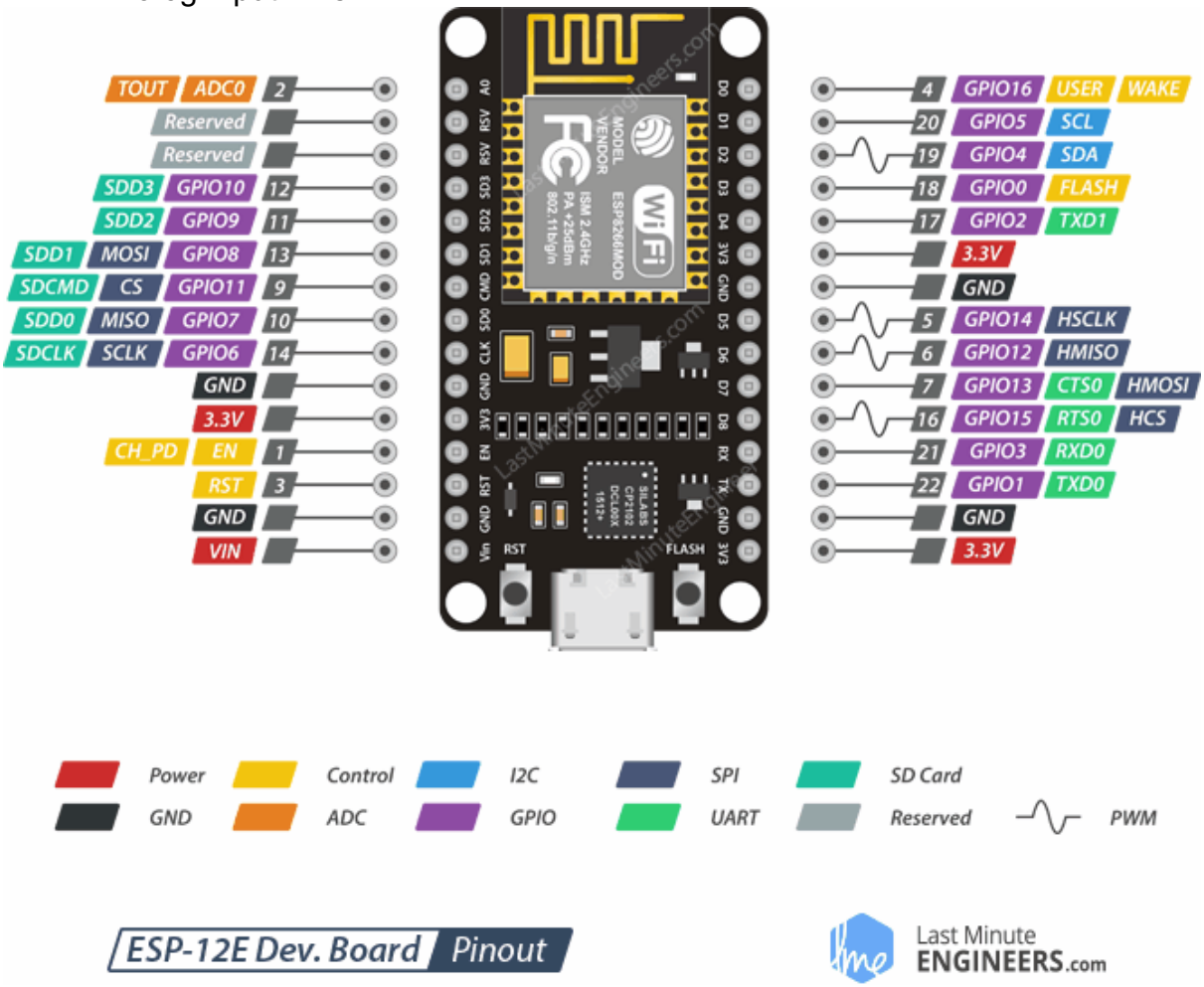
NodeMCU is an open-source IoT platform. It includes firmware that runs on the ESP8266 Wi-Fi SoC from Espressif Systems and hardware based on the ESP-12 module. The ESP8266 chip requires a 3.3V power supply voltage.

NodeMCU ESP-12E dev board can be connected to 5V using a micro USB connector or Vin pin available on the board. The I/O pins of ESP8266 communicate or input/output max 3.3V only, meaning the pins are NOT 5V tolerant inputs.

Key Features:

- Built-in Wi-Fi capability (ESP8266 chip)

- Operating Voltage: 3.3V
- Input Voltage: 5V via USB or Vin
- Digital I/O Pins: 16
- Analog Input Pins: 1

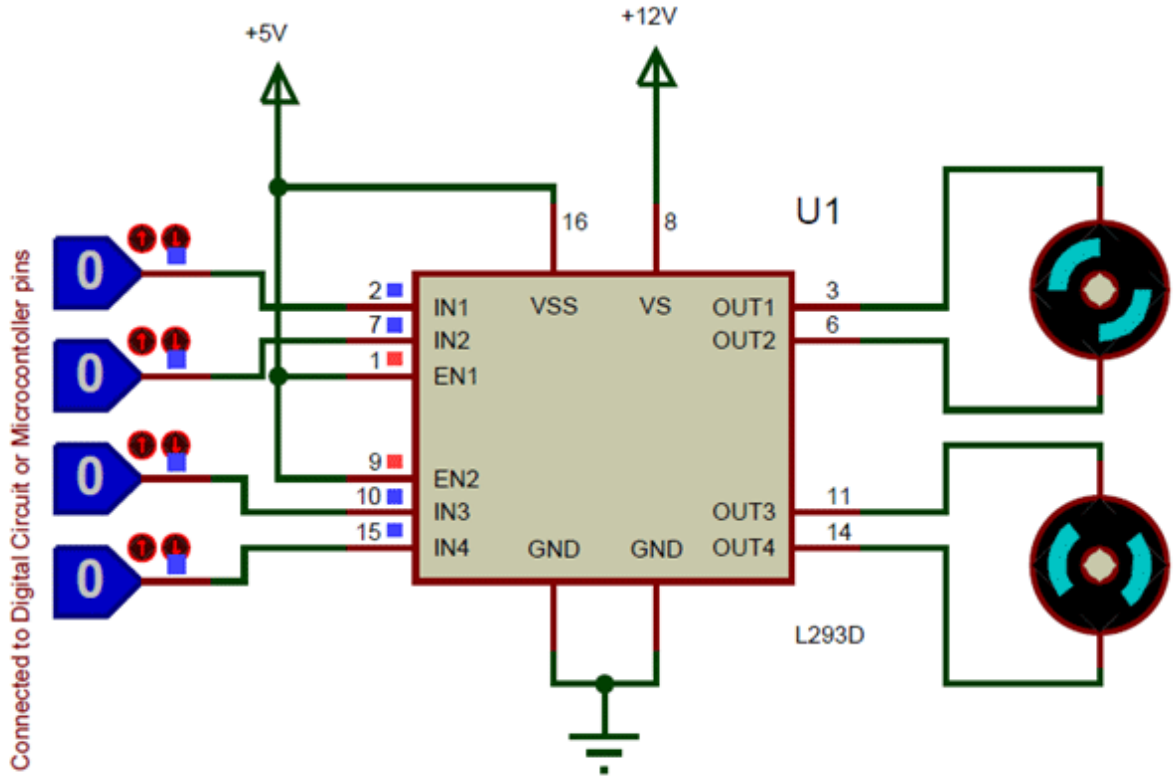


3.2 L293D Motor Driver

The L293D is a popular 16-Pin Motor Driver IC. As the name suggests, it is mainly used to drive motors. A single L293D IC is capable of running two DC motors at the same time; also the direction of these two motors can be controlled independently.

Specifications:

- Motor Voltage (Vcc2/Vs): 4.5V to 36V
- Supply Voltage (Vcc1/VSS): 4.5V to 7V
- Output Current: 600mA per channel
- Peak Current: 1.2A per channel



3.3 BO Motors (Battery Operated)

BO (Battery Operated) motors are lightweight DC geared motors which give good torque and RPM at lower voltages. This motor can run at approximately 150 RPM when driven by a single Li-Ion cell. Great for battery-operated lightweight robots.

DC motors convert electrical energy into mechanical energy. These BO motors allow the robot to move in the desired direction.

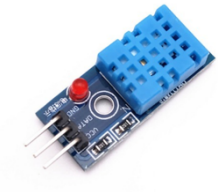
3.4 DHT11 Temperature & Humidity Sensor

The DHT11 is a basic, ultra low-cost digital temperature and humidity sensor. It uses a capacitive humidity sensor and a thermistor to measure the surrounding air and outputs a digital signal on the data pin (no analog input pins are needed).

It's fairly simple to use but requires careful timing to grab data. You can get new data from it once every 2 seconds, so when using the library, sensor readings can be up to 2 seconds old.

Specifications:

- Operating Voltage: 3.3V to 5V
- Temperature Range: 0°C to 50°C
- Humidity Range: 20% to 90% RH
- Sampling Rate: 1 Hz (once every 2 seconds)



3.5 NEO-6M GPS Module

GPS modules contain tiny processors and antennas that directly receive data sent by satellites through dedicated RF frequencies. From there, it receives a timestamp from each visible satellite, along with other pieces of data.

The GPS receiver also knows the exact position in the sky of the satellites at the moment they sent their signals. So given the travel time of the GPS signals from three satellites and their exact position in the sky, the GPS receiver can determine your position in three dimensions – east, north, and altitude.

NEO-6M GPS Module Features:

- Built-in ceramic antenna with strong satellite search capability
- Can track up to 22 satellites on 50 channels
- Sensitivity level: -161 dBm
- Time-To-First-Fix (TTFF): Under 1 second
- Operating Voltage: 2.7V to 3.6V DC
- Operating Current: 67 mA
- Baud Rate: 4800-230400 bps (9600 Default)
- Communication Protocol: NMEA
- Interface: UART
- External antenna and built-in EEPROM

3.6 Power Supply

Lithium-ion batteries are used to give power to the system. They are connected to the motor driver, NodeMCU, and the sensors. Each battery consists of 3.7V and

when connected in series, they will produce approximately 11.1V. This powers all the sensors and actuators present in the system.

3.7 Jumper Wires

A jump wire (also known as jumper wire, or jumper) is an electrical wire, or group of them in a cable, with a connector or pin at each end (or sometimes without them – simply "tinned"), which is normally used to interconnect the components of a breadboard or other prototype or test circuit, internally or with other equipment or components, without soldering.

4. SOFTWARE COMPONENTS

4.1 Arduino IDE

The Arduino Integrated Development Environment (Arduino Software IDE) contains a text editor for writing code, a message area, a text console, a toolbar with buttons for common functions, and a series of menus. It connects to the Arduino/NodeMCU hardware to upload programs and communicate with them.

4.2 ThingSpeak

ThingSpeak™ is an IoT analytics platform service that allows you to aggregate, visualize, and analyze live data streams in the cloud. ThingSpeak provides instant visualizations of data posted by your devices to ThingSpeak. With the ability to execute MATLAB® code in ThingSpeak, you can perform online analysis and processing of the data as it comes in.

4.3 Blynk App

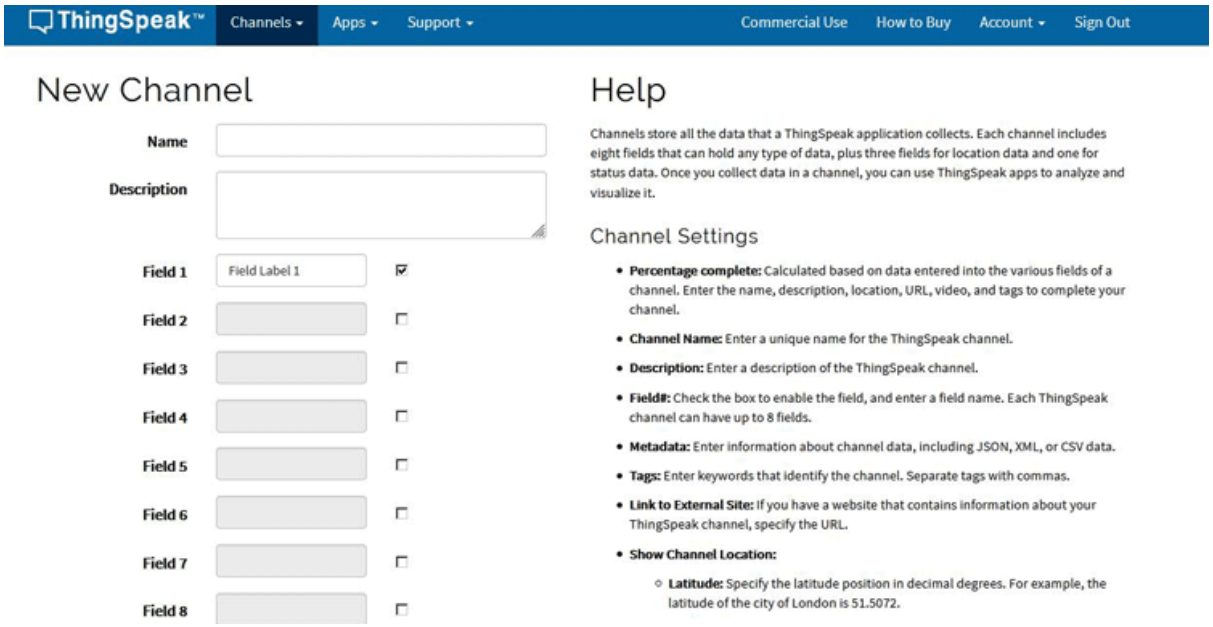
Blynk was designed for the Internet of Things. It can control hardware remotely, display sensor data, store data, visualize it, and do many other cool things. There are three major components in the Blynk platform: Blynk App, Blynk Server, and Blynk Libraries. These components work together to provide real-time IoT device control and monitoring.

5. THINGSPEAK SETUP GUIDE

After successful completion of hardware as per the circuit diagram, it's time to set up the IoT platform where the GPS coordinates are stored. Here we are using ThingSpeak to store the latitude and longitude data on the cloud and graphically visualize the GPS data.

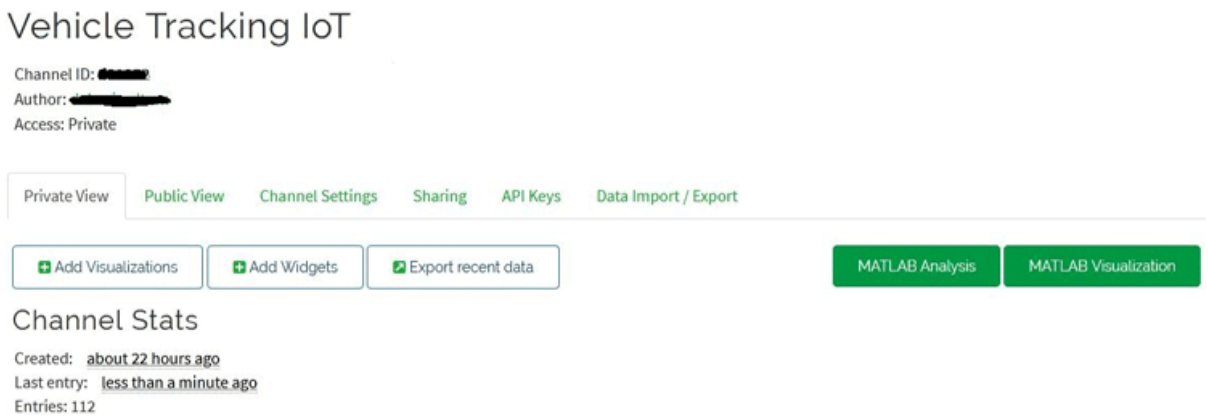
Step 1: Sign up for ThingSpeak

Go to https://thingspeak.com/ and create a new free MathWorks account if you don't have one.



Step 2: Sign in to ThingSpeak

Sign in to ThingSpeak using your credentials and click on "New Channel". Fill up the details of the project like Name, Field names, etc. Create two field names: Latitude and Longitude. Then click on "Save channel".



Step 3: Record the Credentials

Select the created channel and record the following credentials:

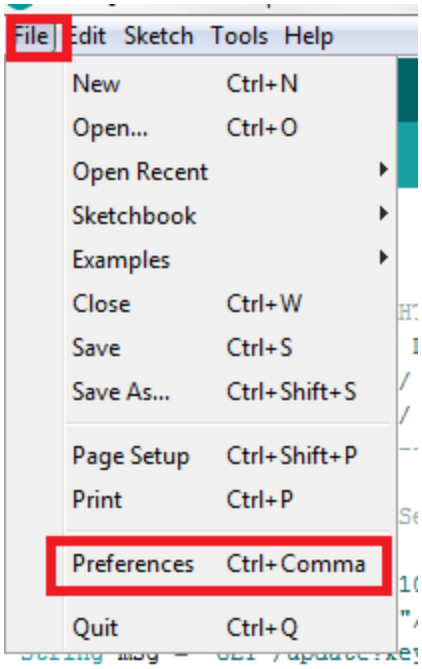
- Channel ID: Found at the top of the channel view
- Write API Key: Found on the API Keys tab of your channel view

6. NODEMCU PROGRAMMING GUIDE

6.1 Setting up Arduino IDE for NodeMCU

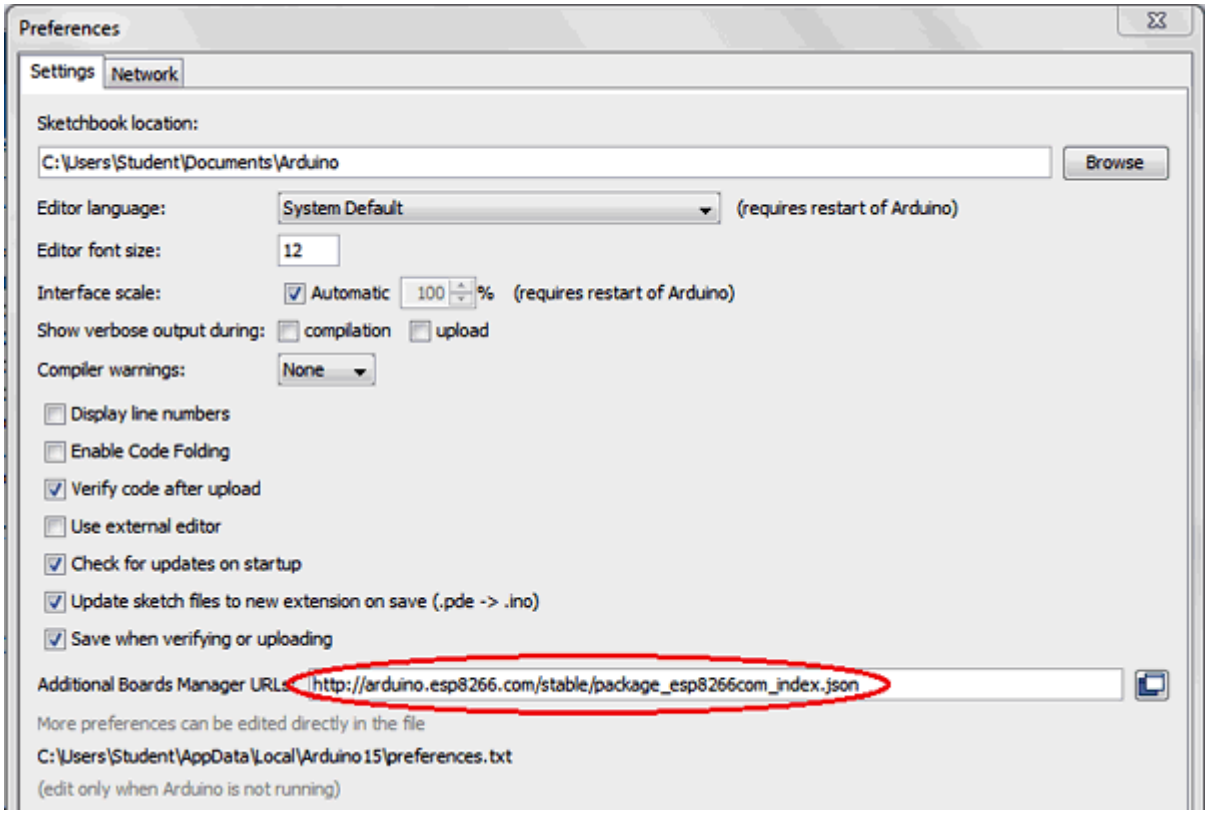
Follow these steps to configure Arduino IDE for NodeMCU programming:

Step 1: Open Arduino IDE, then go to File → Preferences → Settings.



Step 2: Type the following URL in the 'Additional Board Manager URL' field and click 'OK':

https://arduino.esp8266.com/stable/package_esp8266com_index.json



Step 3: Go to Tools > Board > Boards Manager. In the Boards Manager window, type ESP8266 in the search box, select the latest version of the board and click install.

Step 4: After installation is complete, go to Tools → Board → and select NodeMCU 1.0 (ESP-12E Module). Now you can program NodeMCU with Arduino IDE.

6.2 Required Libraries

The following libraries are required for this project:

- **TinyGPS++.h** - For parsing GPS data
- **SoftwareSerial.h** - For serial communication with GPS
- **ThingSpeak.h** - For cloud connectivity
- **ESP8266WiFi.h** - For Wi-Fi connectivity
- **LiquidCrystal_I2C.h** - For LCD display
- **Wire.h** - For I2C communication

After setting up NodeMCU in Arduino IDE, upload the code into NodeMCU. Complete code is given at the end of this tutorial; here we are explaining the code step by step.

Start the code by including all the required library files in the code like *ESP8266WiFi.h* for ESP8266 board, *LiquidCrystal_I2C.h* for LCD, *Wire.h* for I2C communication, etc.

Here, *ThingSpeak.h* library is added to use the ThingSpeak platform with NodeMCU and *TinyGPS++.h The* library is used to get the GPS coordinates using the GPS receiver module. This library can be downloaded from here.

```
#include <TinyGPS++.h>
#include <SoftwareSerial.h>
#include "ThingSpeak.h"
#include <ESP8266WiFi.h>
#include<LiquidCrystal_I2C.h>
#include<Wire.h>
```

For using the I2C module for 16x2 Alphanumeric LCD, configure it using the **LiquidCrystal_I2C** class. Here we have to pass the address, row, and column number which are 0x27, 16, and 2 respectively in our case.

```
LiquidCrystal_I2C lcd(0x27, 16, 2);
```

Now, declare the network credentials- i.e. SSID and password. It is required to connect the NodeMCU to the internet.

```
const char* ssid     = "admin";
const char* password = "12345678";
```

Now, declare the connection pins of the GPS module and it's the default baud rate, which is 9600 in our case.

```
static const int RX= D6, TX= D7;
static const uint32_t GPSBaud = 9600;
```

Next, declare the ThingSpeak account credentials such as the channel number and write API which was recorded earlier.

```
unsigned long ch_no = 9XXXX;
const char * write_api = "ZWWWDXXXXXXXXX";
```

Then declare the objects for *TinyGPSPlus* and *WiFiClient* class. For using *WiFiServer* properties, the *server* object is defined with Port number 80.

```
TinyGPSPlus gps;
WiFiClient  client;
WiFiServer server(80);
SoftwareSerial soft(RX, TX);
```

Inside ***setup(),*** declare all the input pins and output pins. Then print a welcome message on the LCD which will be displayed during the initialization of the project.

```
 Wire.begin(D2, D1);
  lcd.begin(16, 2);
  lcd.init();
  lcd.backlight();
  lcd.setCursor(0, 0);
  lcd.print("   IOT BASED       ");
  lcd.setCursor(0, 1);
  lcd.print("VEHICLE TRACKING   ");
  delay(2000);
  lcd.clear();
```

To connect NodeMCU to the internet, call **WiFi.begin** and pass network SSID and password as its arguments. Check for the successful network connection using *WiFi.status()* and after a successful connection, print a message on LCD with IP address.

```
 WiFi.begin(ssid, password);
  server.begin();
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    lcd.setCursor(0, 0);
    lcd.print("WiFi connecting...          ");
  }
  lcd.setCursor(0, 0);
  lcd.print("WiFi connected          ");
  lcd.setCursor(0, 1);
  lcd.print(WiFi.localIP());
```

Then connect to the ThingSpeak platform, using saved credentials. For this *ThingSpeak.begin* is used.

```
ThingSpeak.begin(client);
```

Inside loop(), ***encode ()*** is used to ensure a valid GPS sentence is received. When **encode ()** returns "true", a valid sentence has just changed the TinyGPS object's internal state. Here two functions namely *displaydata() and displaywebpage()* are called, when *encode()* returns *true*.

```
while (soft.available() > 0)
    if (gps.encode(soft.read()))
    {
      displaydata();
      displaywebpage();
    }
  if (millis() > 5000 && gps.charsProcessed() < 10)
  {
    Serial.println(F("GPS Connection Error!!"));
    while (true);
  }
```

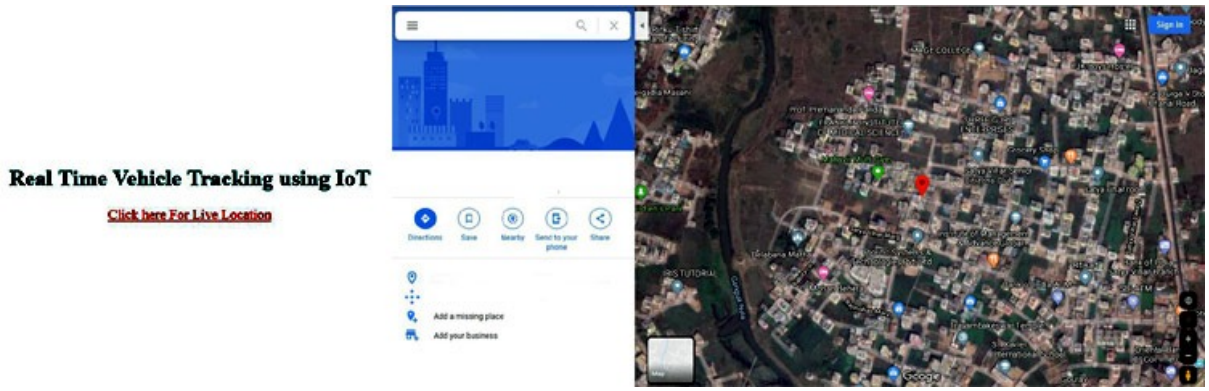
Inside ***displaydata()*** function***,*** *isValid()* method is used to ensure a valid latitude and longitude reception and they are stored in respective variables. Then to send these data to ThingSpeak, *setField()* method is used to set the fields and the *writeFields()* method is used to send these data to the cloud.

```
void displaydata()
{
  if (gps.location.isValid())
  {
    double latitude = (gps.location.lat());
    double longitude = (gps.location.lng());
    latitude_data= (String(latitude, 6));
    longitude_data= (String(longitude, 6));
```

```
    ThingSpeak.setField(1, latitude_data);
    ThingSpeak.setField(2, longitude_data);
    ThingSpeak.writeFields(ch_no, write_api);
    delay(20000);
  }
  else
  {
    Serial.println(F("Data error!!!"));
  }
}
```

Inside *displaywebpage(),* a **HTML code** is written which is sent to Client-side in string format using *client.print().* This HTML code contains a hyperlink which on clicking, takes you to the Google maps pointing the location of a tracked vehicle.

```
{
    WiFiClient client = server.available();
    if (!client)
    {
      return;
    }
    String page = "<html><center><p><h1>Real Time Vehicle
Tracking using IoT</h1><a style=""color:RED;font-size:125%;""
href=""http://maps.google.com/maps?&z=15&mrt=yp&t=k&q=";
    page += latitude_data;
    page += "+";
    page += longitude_data;
    page += ">Click here For Live Location</a>
</p></center></html>";
    client.print(page);
    delay(100);
}
```

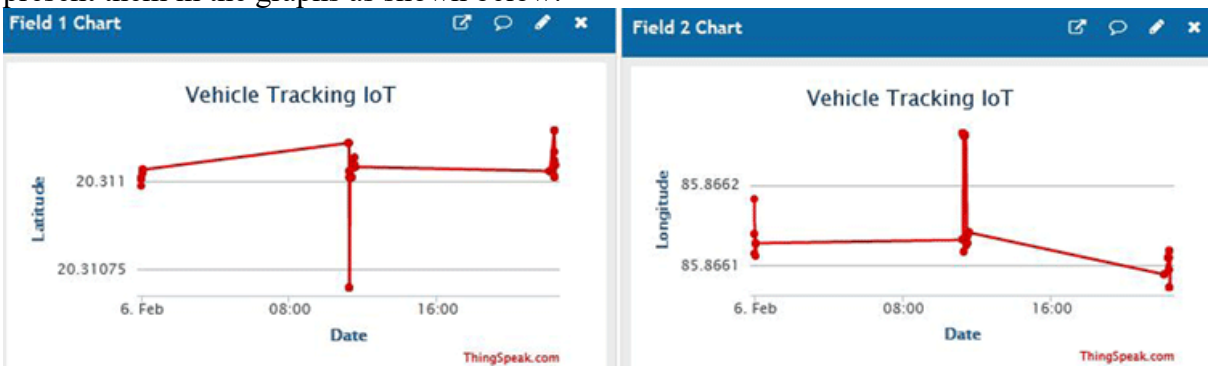


Testing of Vehicle Monitoring System using IoT

After connecting the hardware and uploading the code, just power on the circuit and you will see some notifications messages on LCD. Now open the web browser and open type the IP address of the NodeMCU. There will be a link that will take you to the google map with the current location of the vehicle as shown in the above pictures. The IP address of NodeMCU is

displayed on LCD after Wi-Fi is connected successfully. Complete working is demonstrated in the video given below.

At the same time ThingSpeak will also log the Latitude and Longitude of the vehicle and present them in the graphs as shown below:



7. COMPLETE SOURCE CODE

Below is the complete Arduino code for the IoT Vehicle Tracking System:

```
#include <TinyGPS++.h>
#include <SoftwareSerial.h>
#include "ThingSpeak.h"
#include <ESP8266WiFi.h>
#include <LiquidCrystal_I2C.h>
#include <Wire.h>

LiquidCrystal_I2C lcd(0x27, 16, 2);
static const int RX = D6, TX = D7;
static const uint32_t GPSBaud = 9600;

const char* ssid = "admin";  // Replace with Network SSID
const char* password = "12345678";  // Replace with Network Password

unsigned long ch_no = 12345;  // Replace with ThingSpeak Channel number
const char* write_api = "ZS6XXXXXXXXXXX";  // Replace with ThingSpeak
write API

TinyGPSPlus gps;
WiFiClient client;
WiFiServer server(80);
SoftwareSerial soft(RX, TX);
String latitude_data;
String longitude_data;

void setup()
{
  Wire.begin(D2, D1);
  lcd.begin(16, 2);
  lcd.init();
  lcd.backlight();
  lcd.setCursor(0, 0);
  lcd.print("   IOT BASED       ");
  lcd.setCursor(0, 1);
  lcd.print("VEHICLE TRACKING   ");
  delay(2000);
  lcd.clear();
  Serial.begin(115200);
  soft.begin(GPSBaud);
  WiFi.begin(ssid, password);
  server.begin();
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    lcd.setCursor(0, 0);
    lcd.print("WiFi connecting...          ");
  }
  lcd.setCursor(0, 0);
  lcd.print("WiFi connected          ");
  lcd.setCursor(0, 1);
  lcd.print(WiFi.localIP());
  ThingSpeak.begin(client);
}
void loop()
{
  while (soft.available() > 0)
```

```
    if (gps.encode(soft.read()))
    {
      displaydata();
      displaywebpage();
    }
  if (millis() > 5000 && gps.charsProcessed() < 10)
  {
    Serial.println(F("GPS Connection Error!!"));
    while (true);
  }
}
void displaydata()
{
  if (gps.location.isValid())
  {
    double latitude = (gps.location.lat());
    double longitude = (gps.location.lng());
    latitude_data= (String(latitude, 6));
    longitude_data= (String(longitude, 6));
    ThingSpeak.setField(1, latitude_data);
    ThingSpeak.setField(2, longitude_data);
    ThingSpeak.writeFields(ch_no, write_api);
    delay(20000);
  }
  else
  {
    Serial.println(F("Data error!!!"));
  }
}
void displaywebpage()
{
    WiFiClient client = server.available();
    if (!client)
    {
      return;
    }
    String page = "<html><center><p><h1>Real Time Vehicle Tracking using
IoT</h1><a style=""color:RED;font-size:125%;""
href=""http://maps.google.com/maps?&z=15&mrt=yp&t=k&q=";
    page += latitude_data;
    page += "+";
    page += longitude_data;
    page += ">Click here For Live Location</a> </p></center></html>";

    client.print(page);
    delay(100);
}
```

Note: *The complete code including setup(), loop(), displaydata(), and displaywebpage() functions is available in the accompanying code file.*

8. TESTING AND RESULTS

After connecting the hardware and uploading the code, power on the circuit and you will see notification messages on the LCD. Follow these testing steps:

1. Power on the circuit and observe the LCD for initialization messages
2. Wait for Wi-Fi connection - the IP address will be displayed on the LCD
3. Open a web browser and type the IP address of the NodeMCU

4. Click the link on the webpage to open Google Maps with the current vehicle location
5. Check ThingSpeak to view the logged Latitude and Longitude data in graphical format

9. ADVANTAGES

- Can be used to track the location of a particular thing or vehicle in real-time
- Can be used to measure humidity and temperature at any place
- Can be operated automatically without human intervention
- Reduces manpower requirements for vehicle monitoring
- Cost-effective solution using readily available components
- Cloud-based data storage and visualization
- Remote monitoring capability via web interface

10. FUTURE SCOPE

This prototype can be further developed and enhanced with the following improvements:

- **LoRa Technology Integration:** Implementing LoRa (Long Range) technology to enable signal transmission over much longer distances without relying on cellular or Wi-Fi networks
- **Solar Power:** Adding solar panels for sustainable power supply
- **Additional Sensors:** Including fuel level sensors, speed sensors, and engine diagnostics
- **Mobile App Integration:** Developing a dedicated mobile application for better user experience
- **Geofencing:** Adding geofencing capabilities to alert when the vehicle leaves a designated area
- **Machine Learning:** Implementing predictive maintenance using ML algorithms on collected data

— End of Documentation —