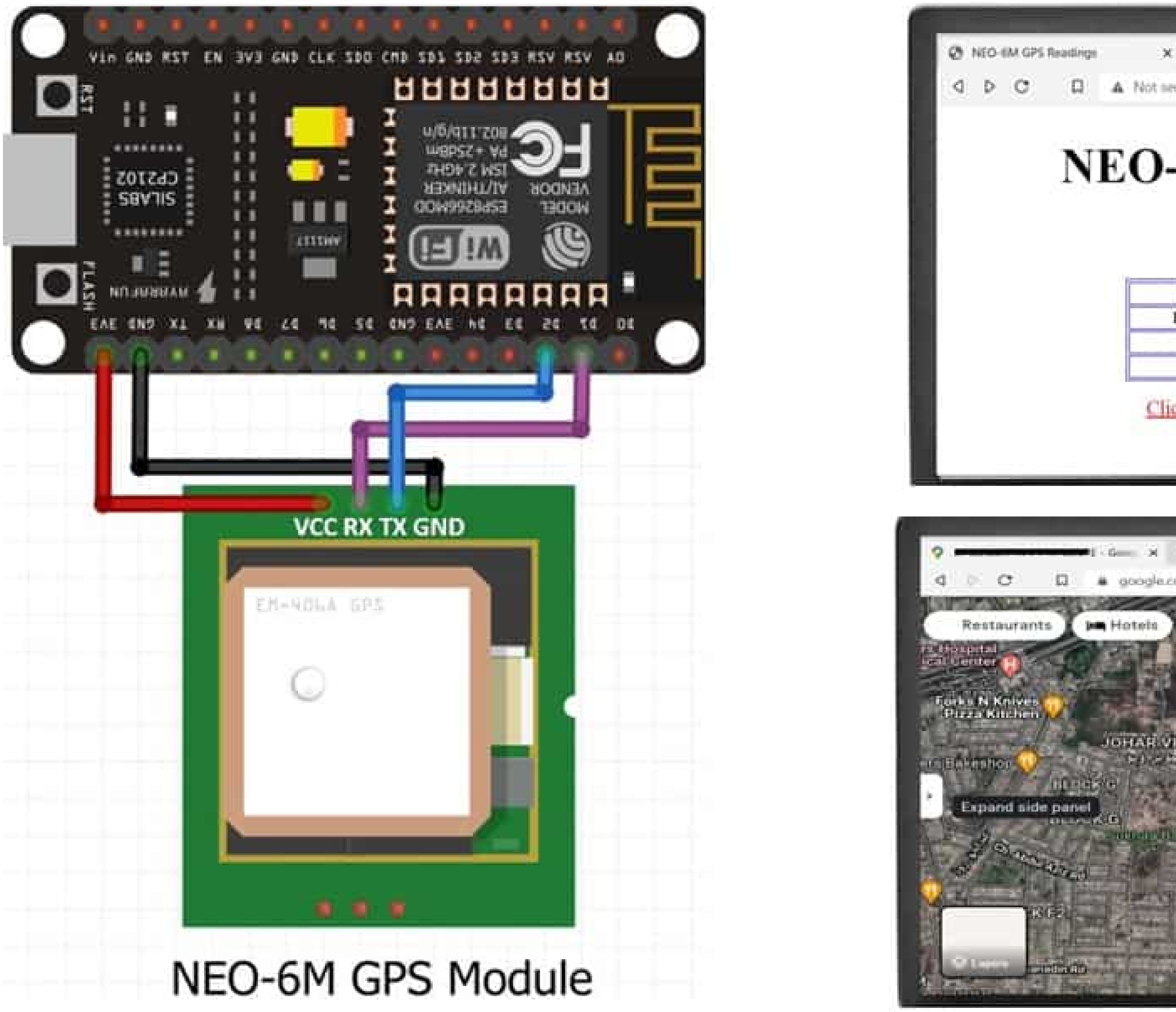
NEO-6M GPS Module with ESP8266 NodeMCU and Track Location on Google Maps

In this tutorial, we will learn about NEO-6M GPS module and how to interface it with ESP8266 NodeMCU. We will learn how GPS works and the overview of the NEO-6M GPS module with an introduction, pinout, and specifications. After that, we will learn to interface a NEO-6M GPS Module module with NodeMCU. We will learn how to obtain GPS parameters such as latitude, longitude, date, time, etc, and display them on a web server, also a user can track location on google maps.


Vin GND RST EN 3V3 GND CLK SDO CMD SD1 SD2 SD3 RSV RSV A0
3V3 GND TX RX D8 D7 D6 D5 GND 3V3 D4 D3 D2 D1 D0
VCC RX TX GND
NEO-6M GPS Module
NEO-6M GPS Readings
NEO-
Restaurants
Hotels
Expand side panel

For demonstration purposes, we will create a web server displaying the GPS parameters acquired from NEO-6M connected with NodeMCU. To make the GPS data readable, we will use the TinyGPS++ library in the sketch. The location will be displayed in the form of longitude and latitude. Additionally, current date and time will also be displayed.

GPS Introduction

Recommended Reading: [GPS (Global Positioning System) – working]

The Global Positioning System (GPS) is a satellite-based navigation system that consists of 24 orbiting satellites, each of which makes two circuits around the Earth every 24 hours. These satellites transmit three bits of information – the satellite's number, its position in space, and the time the information is sent. These signals are picked up by the GPS receiver, which uses this information to calculate the distance between it and the GPS satellites. With signals from three or more satellites, a GPS receiver can triangulate its location on the ground (i.e., longitude and latitude) from the known position of the satellites. With four or more satellites, a GPS receiver can determine a 3D position (i.e., latitude, longitude, and elevation).

In addition, a GPS receiver can provide data on your speed and direction of travel. Anyone with a GPS receiver can access the system. Because GPS provides real-time, three-dimensional positioning, navigation, and timing 24 hours a day, 7 days a week, all over the world, it is used in numerous applications, including GIS data collection, surveying, and mapping.

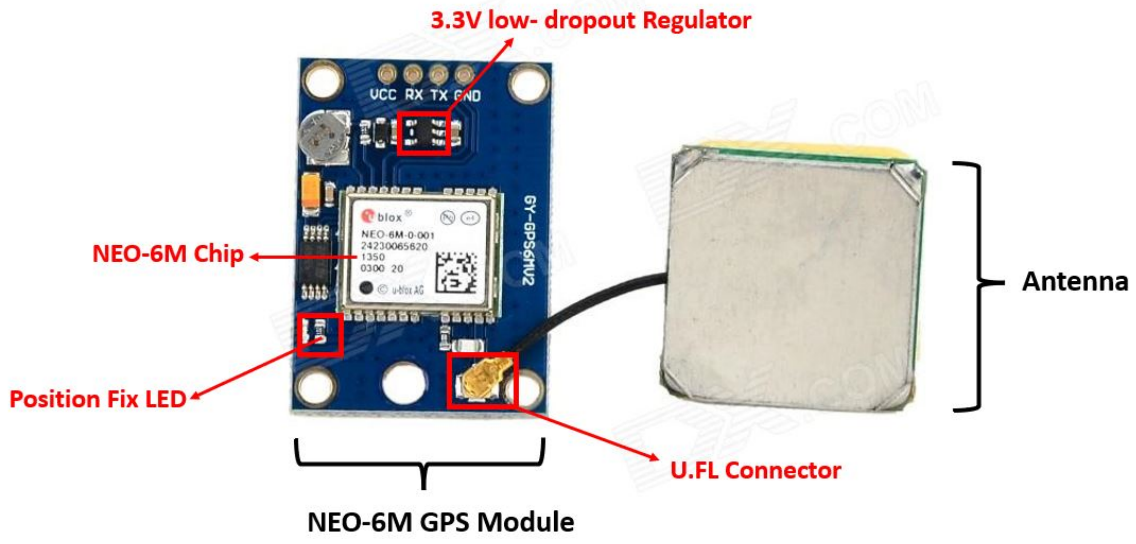
Point to Remember: A GPS receiver locates any three or more of the satellites, calculates the distance to each, and uses this information to generate its own location. This operation is based on a simple mathematical principle called Trilateration.

NEO-6M GPS Module Introduction

The NEO-6M GPS module is a GPS receiver that can locate all locations on Earth as it is able to track approximately 22 satellites. It consists of a high-performance u-blox 6 positioning engine. Measuring 16 x 12.2 x 2.4 mm, its compact architecture along with its low power consumption makes it a good choice for IoT projects. Overall it is a good cost-effective GPS receiver.

Hardware Overview

Let us learn a little bit about its hardware. To obtain GPS readings, we have to use the NEO-6M GPS module with an antenna. The antenna is firmly attached to the module via the U.FL connector. This connector is found on the GPS module.



NEO-6M GPS Chip

In the middle of the GPS module, you can find the NEO-6M chip. This is responsible for tracking up to 22 satellites and any location on the Earth on several channels. Due to its highly sensitive tracking nature, it makes the NEO-6M module a popular GPS tracker.

Some key features of NEO-6M chip include:

- High sensitivity for tracking
- Low supply current (~45mA)
- Is able to track 5 locations per second with an accuracy of 2.5m (horizontal).
- Comes equipped with PSM also known as Power Saving Mode. This mode causes very less power consumption by turning the module ON/OFF according to the need.
- Great use as GPS trackers in smart watches due to very low power consumption (~11mA)

Position Fix LED Indicator

Moving ahead, the module comes with a position fix LED indicator. This LED indicates through its blinking effect whether the module is searching for satellites or has already found them. If the LED blinks after every second, then it indicates that the position fix is found. However, if the LED does not blink then the module is still searching for the satellites.

3.3V low-dropout Regulator

The module also comes equipped with a 3.3V LDO regulator (MIC5205). This provides an efficient linear voltage regulation with ultralow-noise output and very low dropout voltage. Additionally, the module is can also tolerate 5V easily.

Specifications

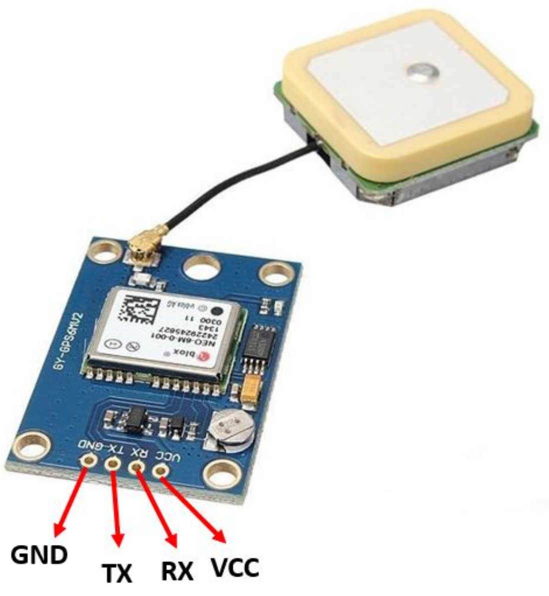
The table below shows some specifications of the NEO-6M module.

Type	GPS
Supply	2.7 V-3.6 V
Operating Current	45mA
Operating Temperature	-40°C ~ 85°C
Horizontal Position Accuracy	2.5m
Communication Protocol	NMEA, UBX Binary, RTCM
Features	RTC Crystal and External Interrupt/Wake up
Interface	UART, SPI, USB and DDC

For more information regarding the NEO-6M module refer to its datasheet given here.

Pinout of NEO 6M Module

The diagram below shows the pinout of the NEO 6M module. It consists of 4 pins named GND, TX, RX, and VCC.



GND	This is the ground pin that will be connected with the ground of ESP8266.
TX	This is the transmission pin used for serial communication.
RX	This is the receiver pin used for serial communication.
VCC	This is the VCC pin used to power up the GPS module. Connect it with the 3.3V of the ESP8266 boa

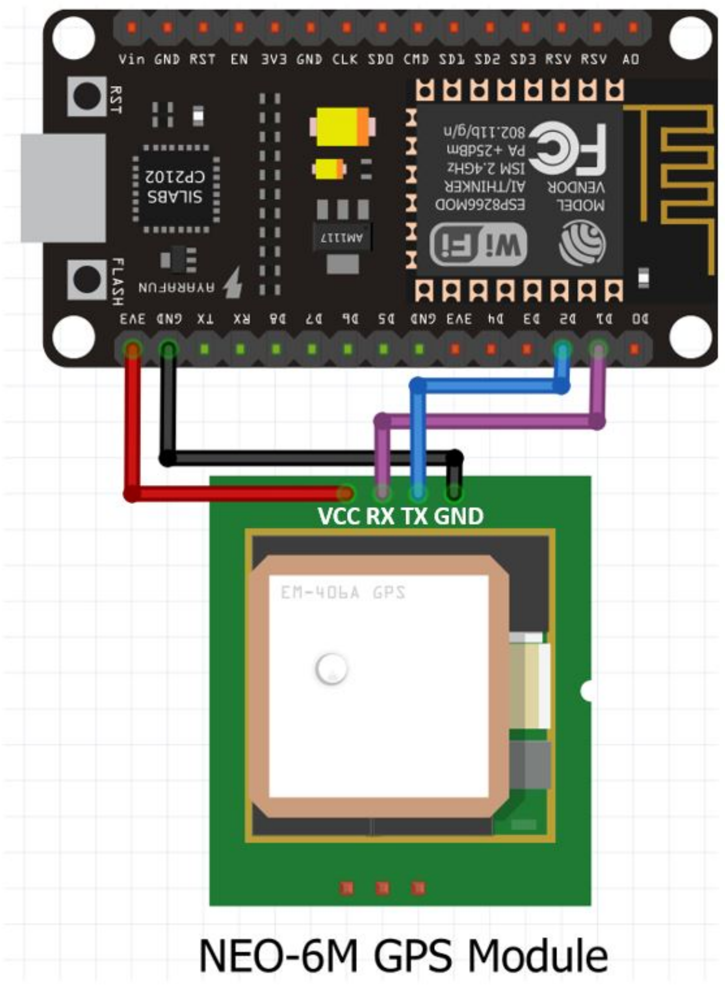
Connection of ESP8266 NodeMCU and NEO-6M Module

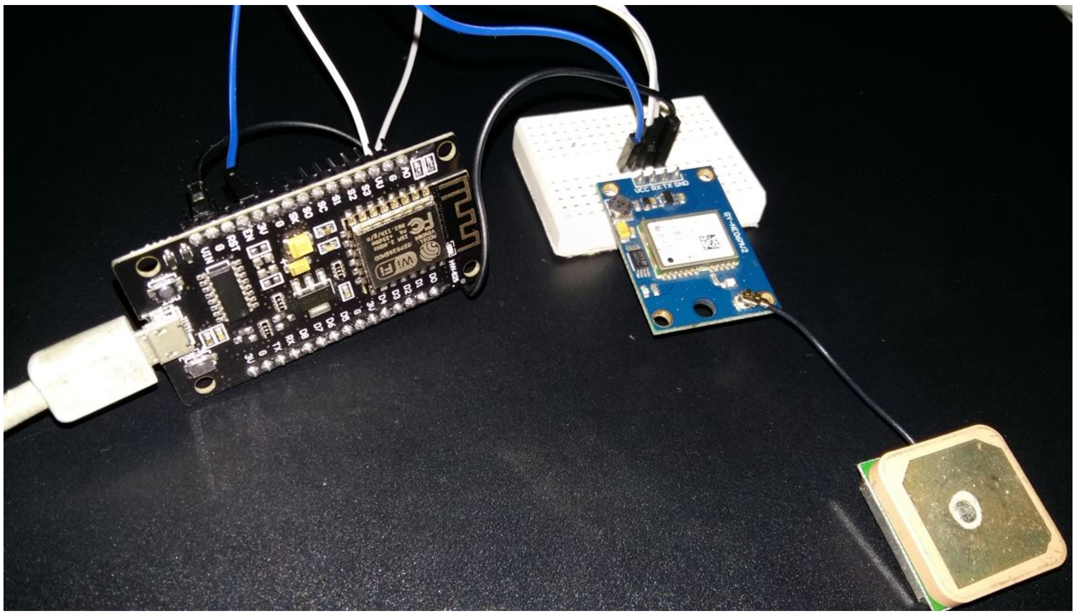
The NEO-6M GPS module has 4 terminals which we will connect with the ESP8266 NodeMCU board. We will connect the VCC terminal with 3.3V pin of ESP8266. We will connect the TX (transmitter) terminal and the RX (receiver) terminal of the GPS module with the GPIO pins of the board. Software serial will be used for communicating between the two devices. Hence, we will set GPIO5 for TX and GPIO4 for RX in the program sketches. Therefore, TX of NEO-6M will be connected with GPIO4 and RX of NEO-6M will be connected with GPIO5. You can use any other GPIO pin as well. Additionally, both the devices will have their grounds in common.

ESP8266	NEO-6M Module
3.3V	VCC
D2 (GPIO4)	TX
D1 (GPIO5)	RX
GND	GND

Schematic Diagram

Connect both the ESP8266 and NEO-6M GPS module as shown below. We have used the same connections as mentioned in the table above.


Vin GND RST EN 3V3 GND CLK SD0 CMD SD1 SD2 SD3 RSV RSV A0
3V3 GND TX RX D8 D7 D6 D5 GND 3V3 D4 D3 D2 D1 D0
VCC RX TX GND
EM-406A GPS
NEO-6M GPS Module
ESP8266 with NEO-6M Module



Understanding basic NMEA sentence

The NEO-6M GPS module sends GPS data in NMEA format. It is of various kinds including $GPRMC, $GPGGA etc. Each NMEA data field is separated by a comma and starts with a '$'.

Below you can view the $GPXXX syntax that denotes the different types of NMEA messages:

$GPGGA	Global Positioning System Fix Data. It provides 3D location and accuracy data
$GPGSA	It provides GPS DOP and active satellites
$GPGSV	It provides the detailed information of the GPS satellite
$GPGLL	It provides the geographic Latitude and Longitude
$GPRMC	It provides the position, velocity and time
$GPVTG	It provides the dual ground/water speed

Let us understand how to read the basic NMEA sentence i.e. $GPGGA. Here is an example $GPGGA NMEA sentence:

$GPGGA, 103005, 3807.038, N, 07128.99030, E, 1, 07, 1.43, 134.5, M, 42.9, M, , *78

- **$**: This indicates the start of the NMEA sentence
- **GPGGA** : Global Positioning System Fix Data
- **103005** : This is the UTC time when the data was accessed in HH:MM:SS. In this case the time is 10:30:05 UTC
- **3807.038, N** : Latitude 38 degrees 07.038′ N
- **07128.99030, E** : Longitude 71 degrees 28.00030′ E
- **1** : GPS fix
- **07** : Number of satellites being tracked
- **1.43** : Horizontal dilution of position
- **134.5, M** : This shows the altitude (m) above the sea level
- **42.9, M** : Height of geoid (mean sea level)
- **Empty field** : time in seconds since last DGPS update
- **Empty field** : DGPS station ID number
- ***78** : the checksum data

Using TinyGPS++ Library to parse NMEA data

We will use Arduino IDE to program our Arduino UNO. To program our NodeMCU to parse NMEA messages easily we will be required to install TinyGPS++.h library.

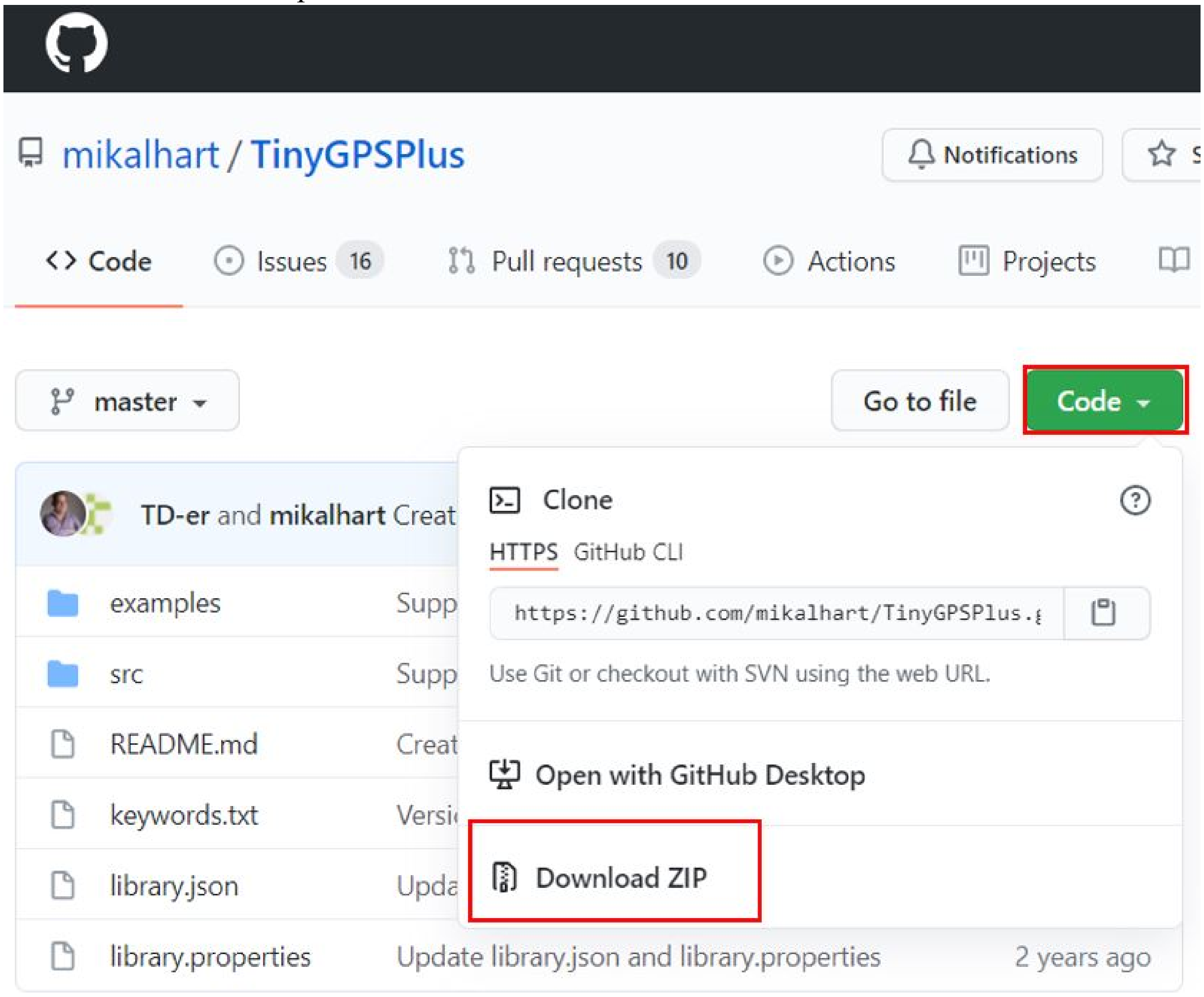
Apart from the functions, we will use in the sketch with TinyGPS++ that will help us access our location’s longitude, latitude as well as the current date and time, we also have some additional functions available by this library:

- gps.speed.value(): This function provides us with the current ground speed in 100ths of a knot.
- gps.course.value(): This function provides us with the current ground course in 100ths of a degree

- gps.hdop.value(): This function provides us with the horizontal dilution of precision (GPS)
- gps.satellites.value(): This function provides us with the number of satellites being tracked
- age(): This method returns the number of milliseconds passed since the last update

Installing TinyGPS++ Library

To make our project easier we will install the TinyGPS++ library to easily handle the receiving data from the GPS module. To download the library, click here. Click on 'Code' and then 'Download Zip'.



You have to go to **Sketch > Include Library > Add .zip Library** inside the IDE to add the library as well.

Arduino Sketch NodeMCU GPS Web Server

Open your Arduino IDE and go to **File > New**. A new file will open. Copy the code given below in that file and save it.
This sketch will display the location's latitude, longitude along with the current date and time in a web sever. This will be achieved through the functions of the TinyGPS++ libray that will easily parse through the NMEA message and display it in a easy convenient format. Moreover we will also include a link to the location in Google Maps in the web server as well.

```
#include <TinyGPS++.h>
#include <SoftwareSerial.h>
#include <ESP8266WiFi.h>

TinyGPSPlus gps;
SoftwareSerial SerialGPS(4, 5);

const char* ssid = "MASTERS";
const char* password = "english123";

float Latitude , Longitude;
int year , month , date, hour , minute , second;
String DateString , TimeString , LatitudeString , LongitudeString;


WiFiServer server(80);
void setup()
{
  Serial.begin(9600);
  SerialGPS.begin(9600);
  Serial.println();
  Serial.print("Connecting");
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");

  server.begin();
  Serial.println("Server started");
  Serial.println(WiFi.localIP());
}

void loop()
{
  while (SerialGPS.available() > 0)
    if (gps.encode(SerialGPS.read()))
    {
      if (gps.location.isValid())
      {
        Latitude = gps.location.lat();
        LatitudeString = String(Latitude , 6);
        Longitude = gps.location.lng();
        LongitudeString = String(Longitude , 6);
      }

      if (gps.date.isValid())
      {
        DateString = "";
        date = gps.date.day();
        month = gps.date.month();
        year = gps.date.year();

        if (date < 10)
        DateString = '0';
        DateString += String(date);

        DateString += " / ";

        if (month < 10)
        DateString += '0';
        DateString += String(month);
        DateString += " / ";

        if (year < 10)
        DateString += '0';
        DateString += String(year);
      }

      if (gps.time.isValid())
      {
        TimeString = "";
        hour = gps.time.hour()+ 5; //adjust UTC
        minute = gps.time.minute();
        second = gps.time.second();

        if (hour < 10)
        TimeString = '0';
        TimeString += String(hour);
        TimeString += " : ";

        if (minute < 10)
        TimeString += '0';
        TimeString += String(minute);
        TimeString += " : ";

        if (second < 10)
        TimeString += '0';
        TimeString += String(second);
      }

    }
  WiFiClient client = server.available();
  if (!client)
  {
    return;
  }

  //Response
  String s = "HTTP/1.1 200 OK\r\nContent-Type: text/html\r\n\r\n <!DOCTYPE
html> <html> <head> <title>NEO-6M GPS Readings</title> <style>";
  s += "table, th, td {border: 1px solid blue;} </style> </head> <body> <h1
style=";
  s += "font-size:300%;";
  s += " ALIGN=CENTER>NEO-6M GPS Readings</h1>";
  s += "<p ALIGN=CENTER style=""font-size:150%;""";
  s += "> <b>Location Details</b></p> <table ALIGN=CENTER style=";
  s += "width:50%";
  s += "> <tr> <th>Latitude</th>";
  s += "<td ALIGN=CENTER >";
  s += LatitudeString;
  s += "</td> </tr> <tr> <th>Longitude</th> <td ALIGN=CENTER >";
  s += LongitudeString;
  s += "</td> </tr> <tr>  <th>Date</th> <td ALIGN=CENTER >";
  s += DateString;
  s += "</td></tr> <tr> <th>Time</th> <td ALIGN=CENTER >";
  s += TimeString;
  s += "</td>  </tr> </table> ";


  if (gps.location.isValid())
  {
    s += "<p align=center><a style=""color:RED;font-size:125%;""
href=""http://maps.google.com/maps?&z=15&mrt=yp&t=k&q=";
    s += LatitudeString;
    s += "+";
    s += LongitudeString;
    s += """ target=""_top"">Click here</a> to open the location in Google
Maps.</p>";
  }

  s += "</body> </html> \n";

  client.print(s);
  delay(100);

}
```

How the Code Works?

Firstly, we will include the libraries which are required for this project. The TinyGPS++ library will be used to extract the useful GPS data whereas the SoftwareSerial library will be used as we are software serial to communicate between the two devices. The ESP8266WiFi.h will be required to connect to the board with the WiFi.

```
#include <TinyGPS++.h>
#include <SoftwareSerial.h>
#include <ESP8266WiFi.h>
```

Then we will set up the RX and TX pins that will be passed as a parameter inside the SoftwareSerial instance called SerialGPS().

```
SoftwareSerial SerialGPS(4, 5);  //RX,TX
```

Additionaly, we will create a TinyGPSPlus object called ‘gps’ which we will use later on in the sketch to acquire the data.

```
TinyGPSPlus gps;
```

Next, we will create two global variables, one for the SSID and another for the password. These will hold our network credentials which will be used to connect to our wireless router. Replace both of them with your network credentials to ensure a successful connection.

```
const char* ssid = "Your_SSID";
const char* password = "Your_Password";
```

Moreover, we will create several variable of type int, float and string to store the GPS parameters.

```
float Latitude , Longitude;
int year , month , date, hour , minute , second;
String DateString , TimeString , LatitudeString , LongitudeString;
```

The WiFiServer object will be used to set up the web server. We will pass the default HTTP port which is 80, as the input to the constructor. This will be the port where the server will listen to the requests.

```
WiFiServer server(80);
```

setup()

Inside the setup() function, we will open the serial communication for both the ESP8266 port and the GPS port at a baud rate of 9600. The NEO-6M has a default baud rate of 9600 so we are using 9600.

```
Serial.begin(9600);
SerialGPS.begin(9600);
```

The following section of code will connect our ESP8266 board with the local network whose network credentials we already specified above. After the connection will be established, the IP address of the board will get printed on the serial monitor. This will help us to make a request to the server.

```
  Serial.println();
  Serial.print("Connecting");
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");

  server.begin();
  Serial.println("Server started");
  Serial.println(WiFi.localIP());
```

loop()

Inside the loop() function, we will first check if GPS data is available and start reading it.

```
 while (SerialGPS.available() > 0)
```

```
    if (gps.encode(SerialGPS.read()))
```
Then we will check if the location is valid or not. For a valid location we will extract the longitude and latitude values and then save them in a string format in their respective variables. The string variables will be used to display the information on the web server.

```
if (gps.location.isValid())
      {
        Latitude = gps.location.lat();
        LatitudeString = String(Latitude , 6);
        Longitude = gps.location.lng();
        LongitudeString = String(Longitude , 6);
      }
```
Likewise we will follow the same procedure and check for a valid date. For a valid date we will extract the date, month and year and then save in a string format in their respective variables.

```
if (gps.date.isValid())
      {
        DateString = "";
        date = gps.date.day();
        month = gps.date.month();
        year = gps.date.year();

        if (date < 10)
        DateString = '0';
        DateString += String(date);

        DateString += " / ";

        if (month < 10)
        DateString += '0';
        DateString += String(month);
        DateString += " / ";

        if (year < 10)
        DateString += '0';
        DateString += String(year);
      }
```
Likewise we will follow the same procedure and check for a valid time. For a valid time we will extract the hour, minute and second and then save it in a string format in their respective variables. Also, we have adjusted the UTC offset as well according to our location. You will have to adjust it yourself.

```
 if (gps.time.isValid())
      {
        TimeString = "";
        hour = gps.time.hour()+ 5; //adjust UTC
        minute = gps.time.minute();
        second = gps.time.second();

        if (hour < 10)
        TimeString = '0';
        TimeString += String(hour);
        TimeString += " : ";

        if (minute < 10)
        TimeString += '0';
        TimeString += String(minute);
        TimeString += " : ";

        if (second < 10)
        TimeString += '0';
        TimeString += String(second);
      }
```

The client gets connected to the sever.

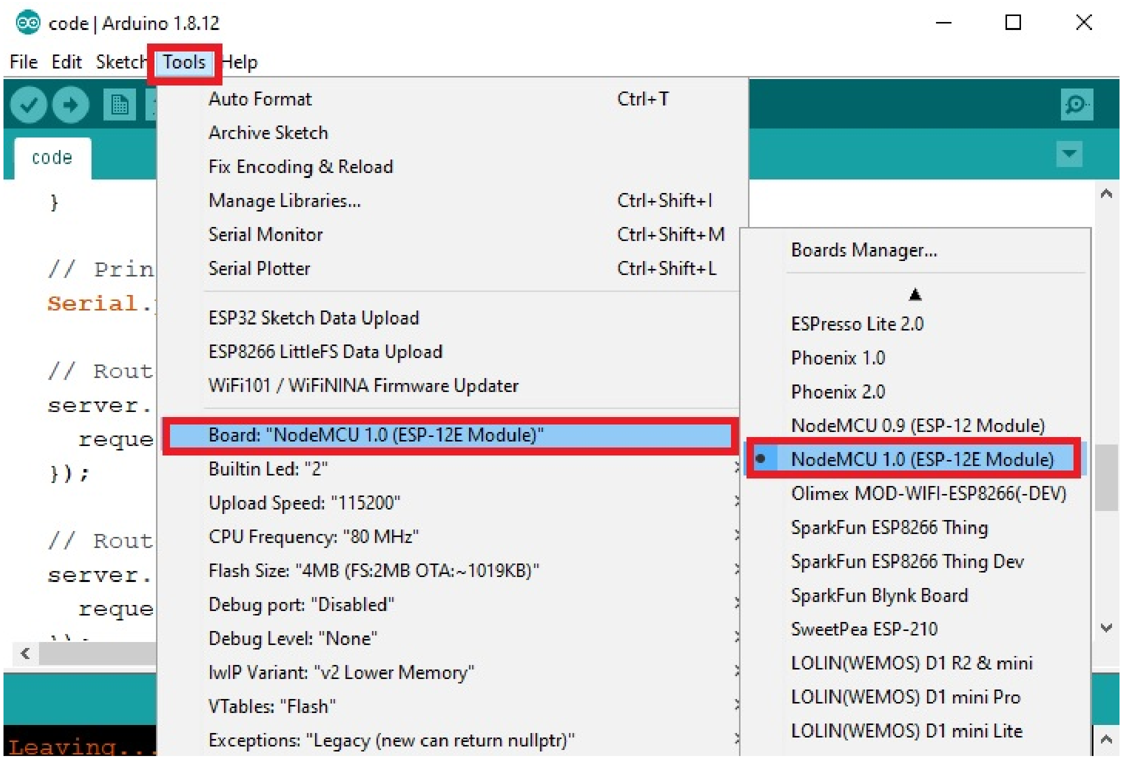
```
 WiFiClient client = server.available();
  if (!client)
  {
    return;
  }
```

This is the response that is sent to the client. Here ‘s’ is a string which contains HTML code for web server along with the GPS parameters. We have created a table to display the Latitude, Longitude, Date and Time values on the web server. These will be received by the NEO-6M GPS module.

```
 //Response
  String s = "HTTP/1.1 200 OK\r\nContent-Type: text/html\r\n\r\n <!DOCTYPE
html> <html> <head> <title>NEO-6M GPS Readings</title> <style>";
  s += "table, th, td {border: 1px solid blue;} </style> </head> <body> <h1
style=";
  s += "font-size:300%;";
  s += " ALIGN=CENTER>NEO-6M GPS Readings</h1>";
  s += "<p ALIGN=CENTER style=""font-size:150%;""";
  s += "> <b>Location Details</b></p> <table ALIGN=CENTER style=";
  s += "width:50%";
  s += "> <tr> <th>Latitude</th>";
  s += "<td ALIGN=CENTER >";
  s += LatitudeString;
  s += "</td> </tr> <tr> <th>Longitude</th> <td ALIGN=CENTER >";
  s += LongitudeString;
  s += "</td> </tr> <tr>  <th>Date</th> <td ALIGN=CENTER >";
  s += DateString;
  s += "</td></tr> <tr> <th>Time</th> <td ALIGN=CENTER >";
  s += TimeString;
  s += "</td>  </tr> </table> ";


  if (gps.location.isValid())
  {
    s += "<p align=center><a style=""color:RED;font-size:125%;""
href=""http://maps.google.com/maps?&z=15&mrt=yp&t=k&q=";
    s += LatitudeString;
    s += "+";
    s += LongitudeString;
    s += """ target=""_top"">Click here</a> to open the location in Google
Maps.</p>";
  }

  s += "</body> </html> \n";

  client.print(s);
  delay(100);
```

Demonstration

To see the demonstration of the above code, upload the code to your board. But, before uploading code, make sure to select NodeMCU 1.0 from Tools > Board.



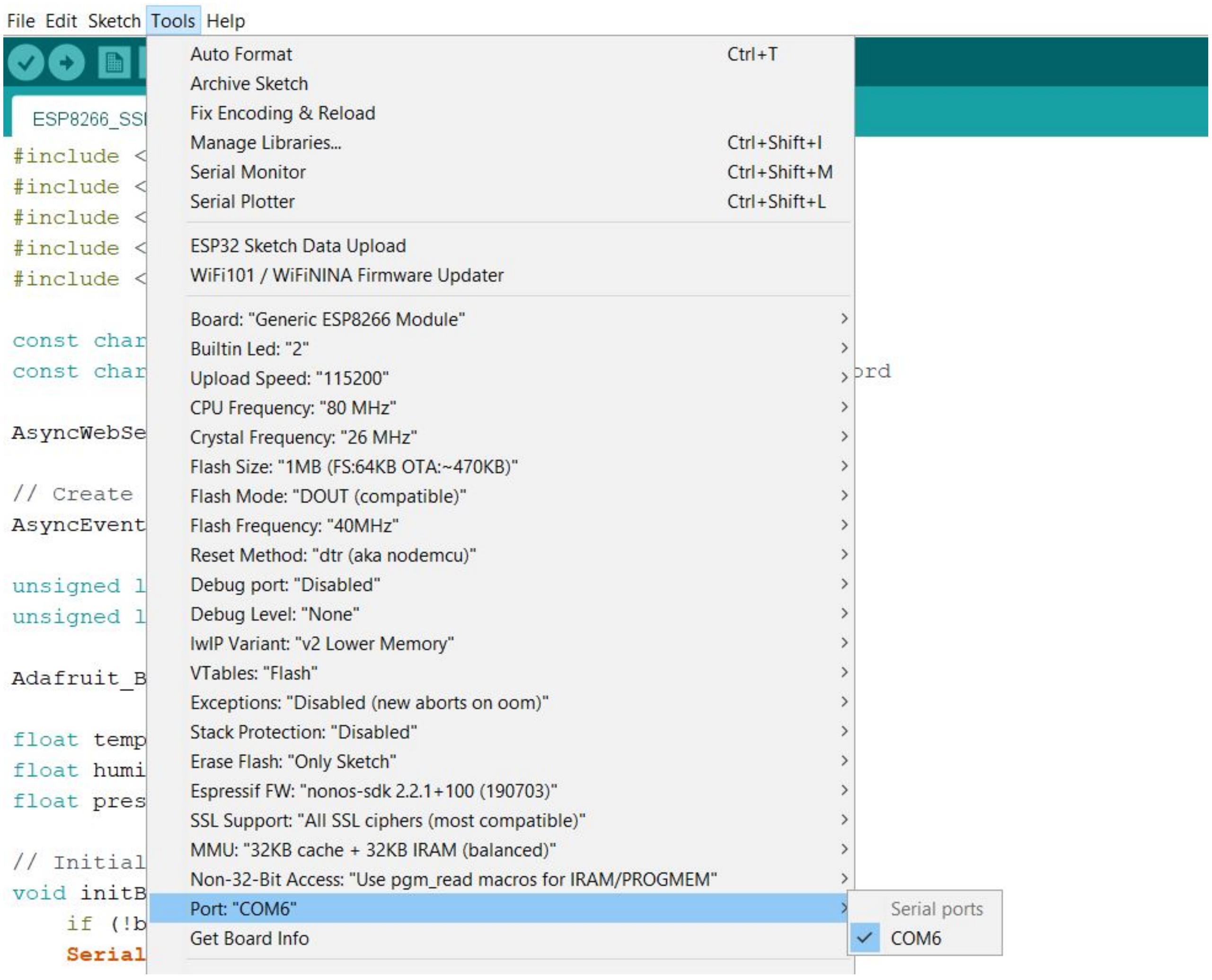
code | Arduino 1.8.12

File Edit Sketch Tools Help

Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
ESP32 Sketch Data Upload
ESP8266 LittleFS Data Upload
WiFi101 / WiFiNINA Firmware Updater
Board: "NodeMCU 1.0 (ESP-12E Module)"
Builtin Led: "2"
Upload Speed: "115200"
CPU Frequency: "80 MHz"
Flash Size: "4MB (FS:2MB OTA:~1019KB)"
Debug port: "Disabled"
Debug Level: "None"
IwIP Variant: "v2 Lower Memory"
VTables: "Flash"
Exceptions: "Legacy (new can return nullptr)"

Boards Manager...
ESPresso Lite 2.0
Phoenix 1.0
Phoenix 2.0
NodeMCU 0.9 (ESP-12 Module)
● NodeMCU 1.0 (ESP-12E Module)
Olimex MOD-WIFI-ESP8266(-DEV)
SparkFun ESP8266 Thing
SparkFun ESP8266 Thing Dev
SparkFun Blynk Board
SweetPea ESP-210
LOLIN(WEMOS) D1 R2 & mini
LOLIN(WEMOS) D1 mini Pro
LOLIN(WEMOS) D1 mini Lite

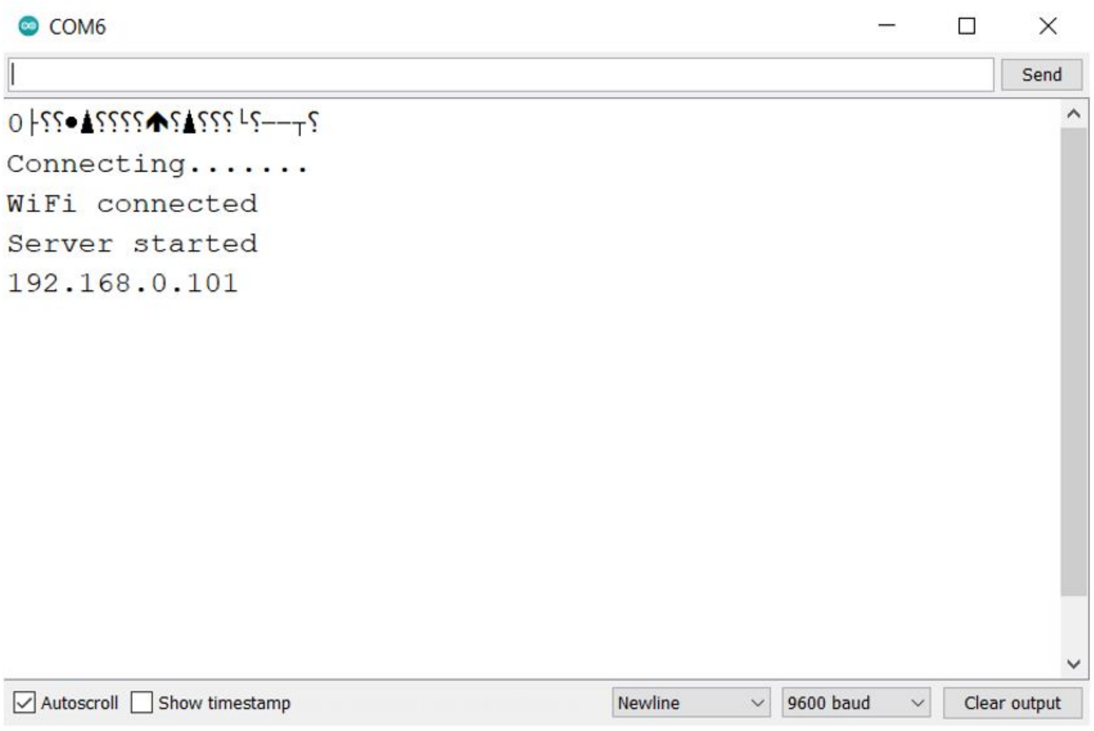
Also select the correct COM port to which the board is connected from Tools > Port.



```
#include <
#include <
#include <
#include <
#include <

const char
const char                                                    ord

AsyncWebSe

// Create
AsyncEvent

unsigned l
unsigned l

Adafruit_B

float temp
float humi
float pres

// Initial
void initB
    if (!b
    Serial
```

Board: "Generic ESP8266 Module"
Builtin Led: "2"
Upload Speed: "115200"
CPU Frequency: "80 MHz"
Crystal Frequency: "26 MHz"
Flash Size: "1MB (FS:64KB OTA:~470KB)"
Flash Mode: "DOUT (compatible)"
Flash Frequency: "40MHz"
Reset Method: "dtr (aka nodemcu)"
Debug port: "Disabled"
Debug Level: "None"
lwIP Variant: "v2 Lower Memory"
VTables: "Flash"
Exceptions: "Disabled (new aborts on oom)"
Stack Protection: "Disabled"
Erase Flash: "Only Sketch"
Espressif FW: "nonos-sdk 2.2.1+100 (190703)"
SSL Support: "All SSL ciphers (most compatible)"
MMU: "32KB cache + 32KB IRAM (balanced)"
Non-32-Bit Access: "Use pgm_read macros for IRAM/PROGMEM"
Port: "COM6"
✓ COM6

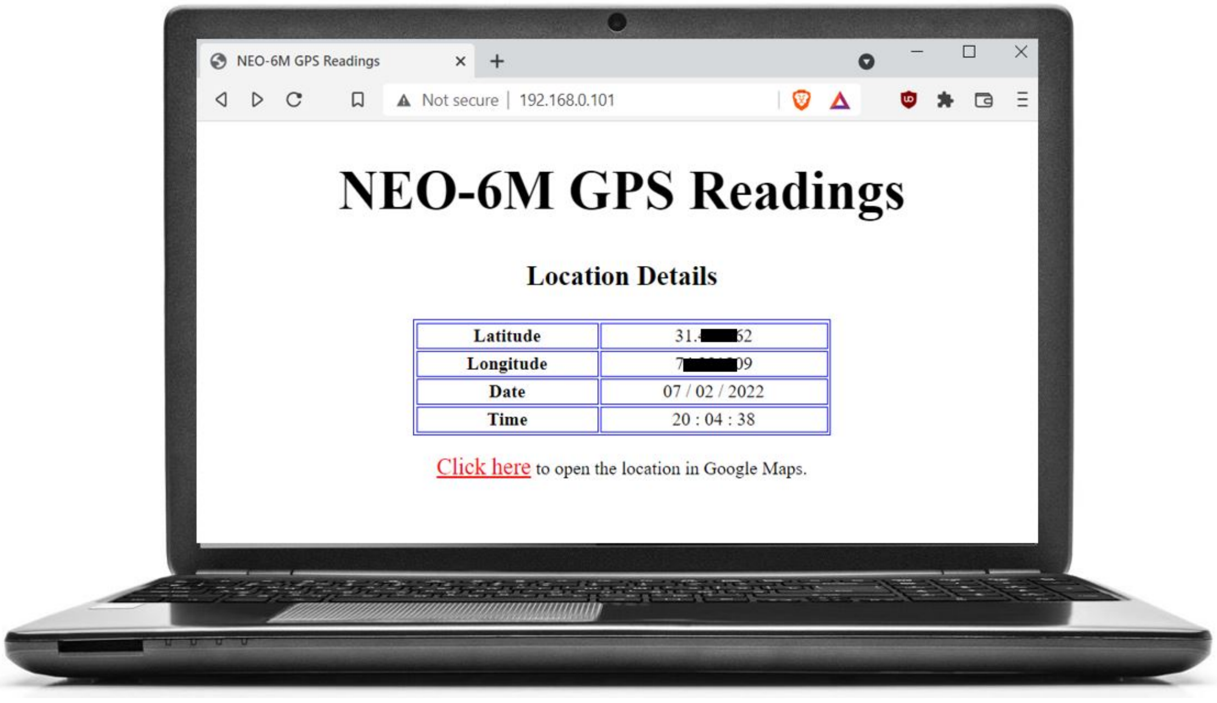
Once the code is uploaded to NodeMCU, open the serial monitor and set the baud rate to 9600. You will be able to see the IP address of your ESP8266 module after it connects with the WiFi.



```
0├ʕʕ•♟ʕʕʕʕ♠ʕ♟ʕʕʕ└ʕ──┬ʕ
Connecting.......
WiFi connected
Server started
192.168.0.101
```

Serial Monitor

Now type this IP address in a new web browser and press enter. The web server will open up. It will list the GPS parameters including longitude, latitude, current date and current time (24 hour) as shown below.



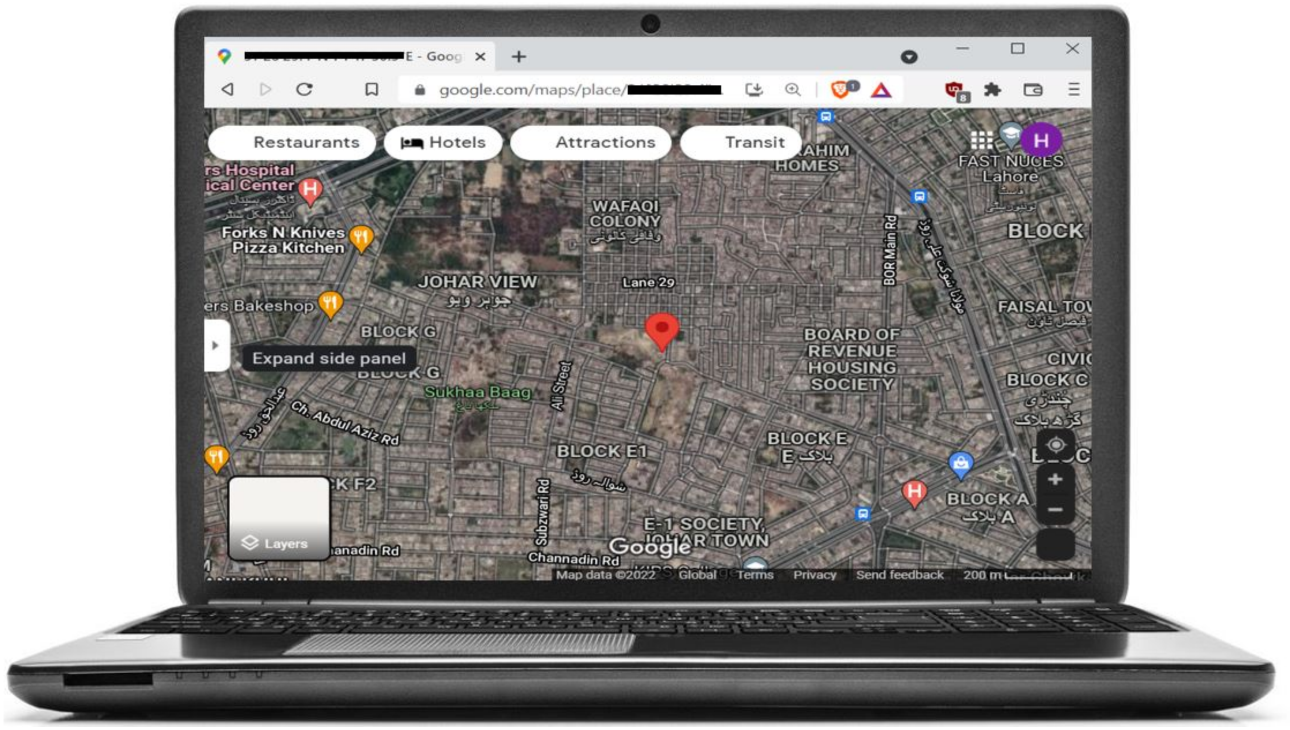
NEO-6M GPS Readings

Location Details

Latitude	31.[redacted]62
Longitude	7[redacted]09
Date	07 / 02 / 2022
Time	20 : 04 : 38

Click here to open the location in Google Maps.

If you click the ‘Click here’ link, Google Map will open up showing the location.



To read more GPS related articles, follow the links below:

- [ESP32 GPS tracker – IoT based Vehicle Tracking System]()
- [Display GPS Co-ordinates on LCD using pic microcontroller]()
- [Interfacing GPS module with Arduino: GPS coordinates on Lcd]()
- [Vehicle Tracking System Through GPS-GSM Modules]()
- [GPS (Global Positioning System) – working]()