



Assembly of circuit with Gas sensor for car and sending IoT data

Difficulty level: Medium

Goals

Automotive IoT is the integration of gadgets, sensors, cloud computing, applications, and other such components into vehicles to function as a complex system for the connection of cars, predictive maintenance, fleet management, OEMs, insurance, and more. The integration of the Internet of Things in the automotive industry allows manufacturers to implement sought-after innovations that can ultimately transform cars into near-artificial intelligence. At a didactic level, we are now going to develop some exercises using sensors for data acquisition, processed by the Arduino microcontroller.

This exercise intends to apply a sensor to detect gases in the atmosphere guided by a microcontroller to alert the occupants if the surrounding atmosphere has little amount of carbon dioxide (CO₂) for example. This way, it is possible to protect passengers from heavy atmospheres, as is the case in some car parks. This exercise is aided by a liquid crystal display (LCD) that presents written information about the amounts of CO₂. For the possible sending of data, it will be necessary to apply, for example, the ESP8266 ESP-01 module that allows the connection of several devices to the internet (or local network), and consequent sending of data from the sensors applied to the autonomous system.

Image-1: Understanding the application of Gas sensor in a car and communicating with IoT.

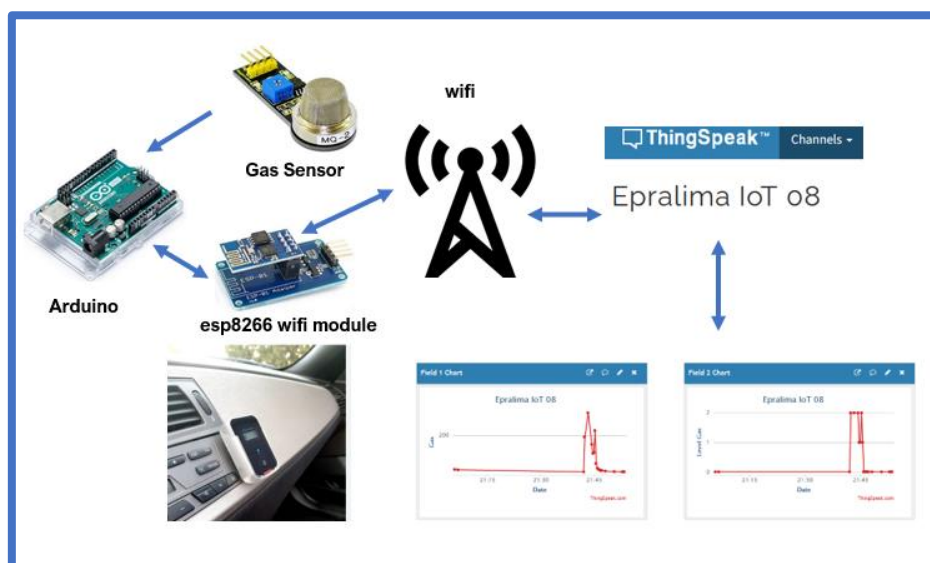


Image 1: application of Gas sensor in a car and communicating with IoT



Skills

- The skills our students will gain are:
- Students' ability to build circuits will be developed.
- The ability to program the Arduino board and use the ESP8266 Module for Internet access will develop.
- The ability to receive data from the brightness sensor and send the received data to Thing Speak will be gained.
- Data analytics will improve their ability to connect with the Internet of Things.

Required materials and circuit diagram.

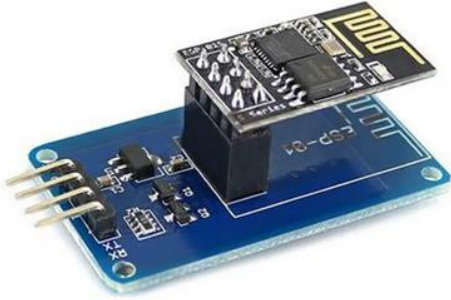
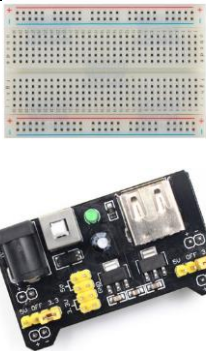


In this exercise we intend to learn how to draw diagrams (circuits), connect all the components correctly, develop software based on C language (Arduino), connect to the wifi network, communicate with an IoT server, ThingSpeak and read server-generated graphics.

Quantity	Component
1	Arduino Uno R3
1	ESP01-8266
1	Power Supply (braedBoard)
1	BreadBoard
1	LCD display 2 x 16 (I2C)
1	MQ-2 Gas Sensor
3	Led Green, Red and Blue
3	Resistor 330Ohm

Table 1 - Components List



Materials table

	
Arduino	ESP01 - 8266
	
Bread Board + Power Supply	MQ-2 Gas Sensor
	
330 Ω Leds	LCD display 2 x 16 (I2C)
	
Jumper wire	

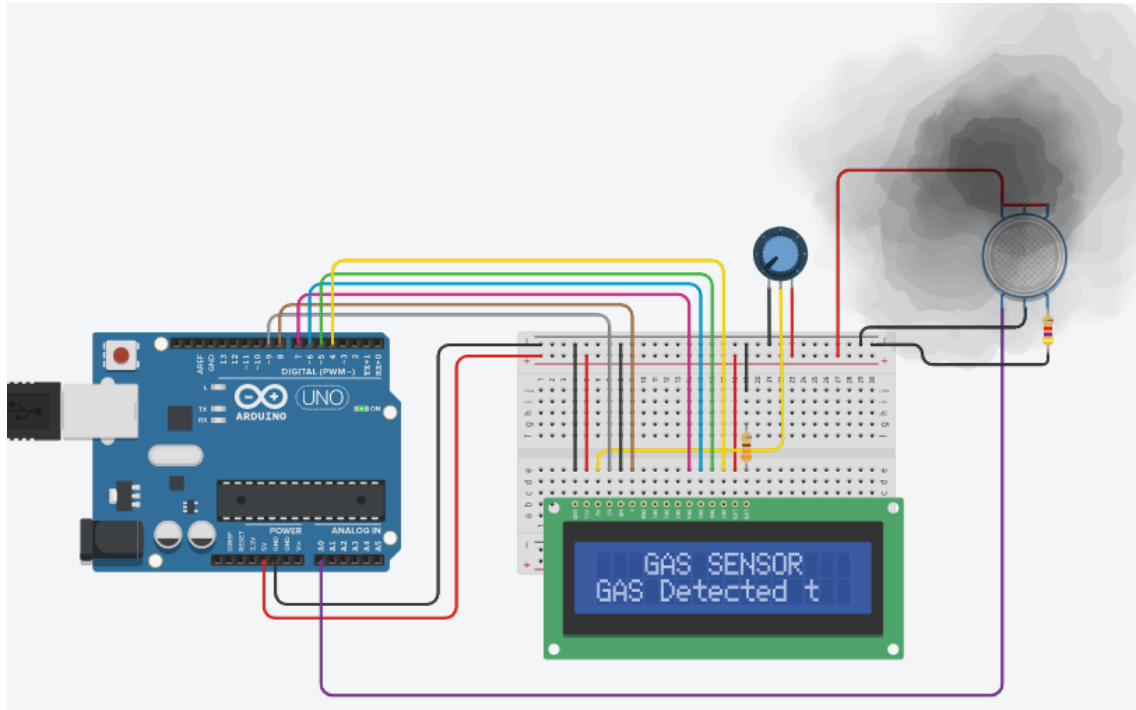


Image 2 – Diagram circuit

Implementation

Development of communication of microcontroller systems, and sensors, with the ThingSpeak IoT cloud.

The ESP8266 WiFi module (image 3) is a small shield with integrated TCP/IP protocol that can give any microcontroller access to the WiFi network. The ESP8266 is capable of both hosting an application and offloading all WiFi network functions from another application processor. Each ESP8266 module is pre-programmed with an AT command making its firmware settings, meaning that we can simply connect this module to the Arduino working as any other WiFi shield would. This module has a great cost/benefit ratio and has a very large and constantly growing user community.

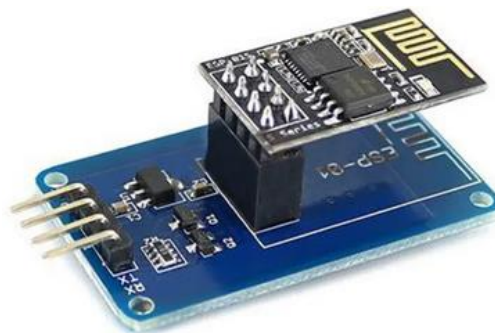


Image 3 - ESP01 – 8266



This analogy gas sensor - MQ2 is used in gas leakage detecting equipment in consumer electronics and industrial markets.

This sensor is suitable for detecting LPG, I-butane, propane, methane, alcohol, Hydrogen, and smoke. It has high sensitivity and quick response.

In addition, the sensitivity can be adjusted by rotating the potentiometer.



Image 4 MQ-2 Gas Sensor

Implementation in practice

1. Assemble the circuit in the image 2;
2. Connect correctly ESP01-8266 image 5

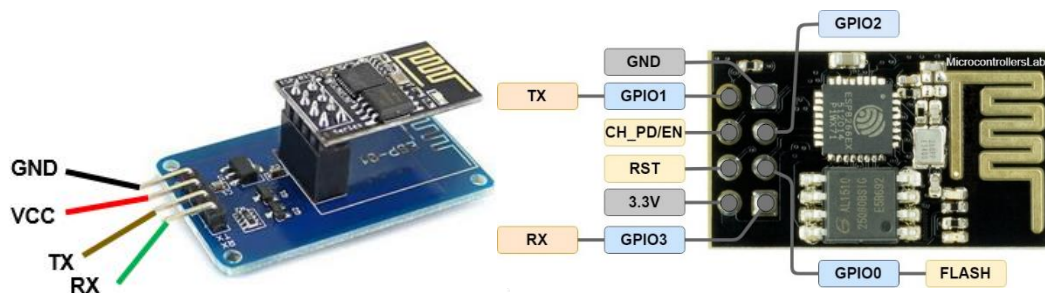


Image 5 ESP-01 Connections



3. Real assembled circuit image 6

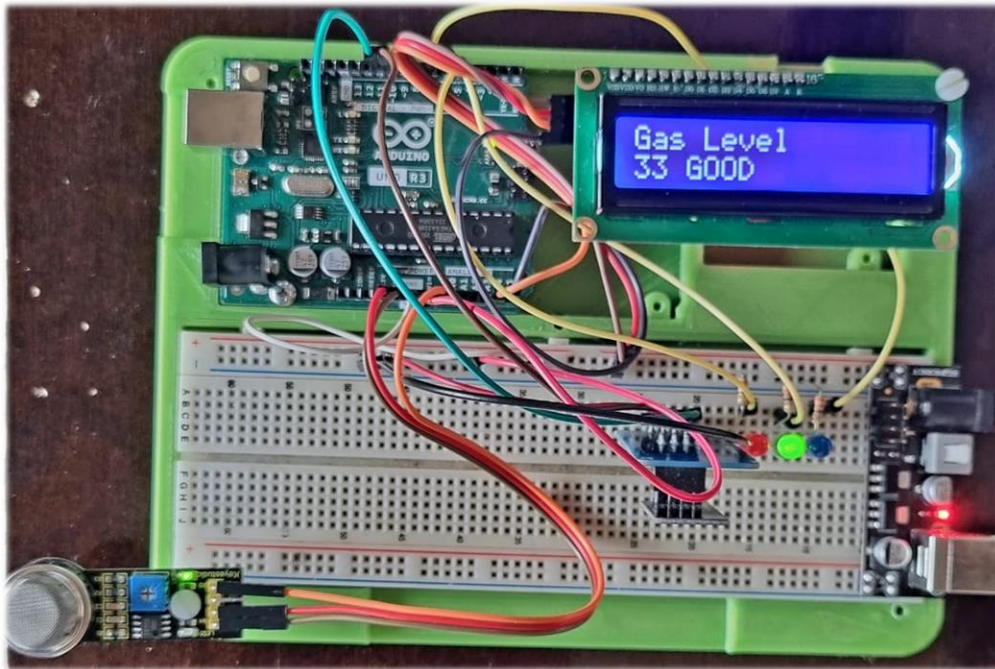


Image 6 Real circuit in breadboard

4. Create a ThingSpeak account image 7

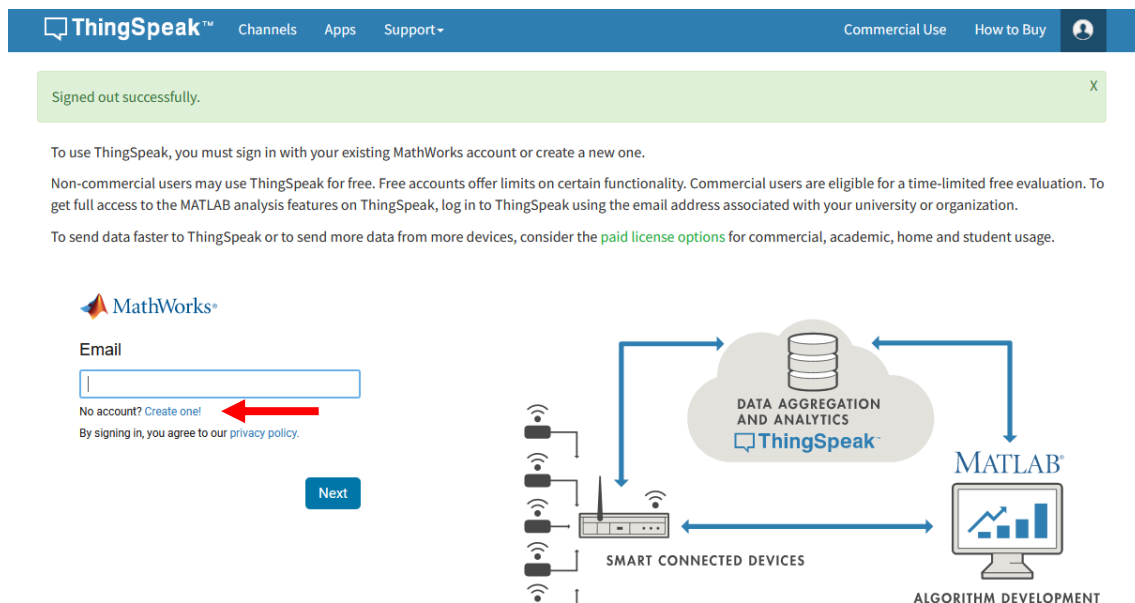


Image 7 - Thing Speak

5. Create a new channel image 8



My Channels

[New Channel](#)

Help

Collect data in a ThingSpeak channel from a device, from another channel, or from the web.

Image 8 Interface ThingSpeak

6. Configure channel, with name, Description, and fields. Image 9.

Note: The fields refer to data processed by the microcontroller and data from the sensors under study. Each field will generate a graph.

ThingSpeak™ Channels Apps Devices Support Commercial Use How to Buy AC

Epralima IoT 01

Channel ID: 2064208
Author: mwa0000029483576
Access: Private

This Channel presents the simulation of car night lighting based on the level and intensity of natural light

[Private View](#) [Public View](#) [Channel Settings](#) [Sharing](#) [API Keys](#) [Data Import / Export](#)

Channel Settings

Percentage complete 50%

Channel ID 2064208

Name

Description

Field 1 ☒

Field 2 ☒

Help

Channels store all the data that a ThingSpeak application collects. Each channel includes eight fields that can hold any type of data, plus three fields for location data and one for status data. Once you collect data in a channel, you can use ThingSpeak apps to analyze and visualize it.

Channel Settings

- Percentage complete:** Calculated based on data entered into the various fields of a channel. Enter the name, description, location, URL, video, and tags to complete your channel.
- Channel Name:** Enter a unique name for the ThingSpeak channel.
- Description:** Enter a description of the ThingSpeak channel.
- Field#:** Check the box to enable the field, and enter a field name. Each ThingSpeak channel can have up to 8 fields.

Image 9 Configure Channel

7. Save settings channel Image 10



ThingSpeak™ Channels Apps Devices Support Commercial Use How to Buy AC

station that acquires data from an Arduino® device. [Learn More](#)

Link to GitHub

Elevation

Show Channel Location ☐

Latitude

Longitude

Show Video ☐

☒ YouTube ☐ Vimeo

Video URL

Show Status ☐

[Save Channel](#)

Image 10 Save settings channel

8. In this step, we will pay special attention to the api keys, as they are the ones that, through the string key, will allow access to the IoT repository in Arduino programming. Also very important are the API requests.

ThingSpeak™ Channels Apps Devices Support Commercial Use How to Buy AC

Private View Public View Channel Settings Sharing API Keys Data Import / Export

Write API Key

[Key](#)

[Generate New Write API Key](#)

Read API Keys

[Key](#)

Note

[Save Note](#) [Delete API Key](#)

Help

API keys enable you to write data to a channel or read data from a private channel. API keys are auto-generated when you create a new channel.

API Keys Settings

- **Write API Key:** Use this key to write data to a channel. If you feel your key has been compromised, click **Generate New Write API Key**.
- **Read API Keys:** Use this key to allow other people to view your private channel feeds and charts. Click **Generate New Read API Key** to generate an additional read key for the channel.
- **Note:** Use this field to enter information about channel read keys. For example, add notes to keep track of users with access to your channel.

API Requests

Write a Channel Feed

```
GET https://api.thingspeak.com/update?api_key=UC[redacted]&field1=< >
```

Read a Channel Feed

```
GET https://api.thingspeak.com/channels/[redacted]/feeds.json?api_key=D8[redacted]< >
```

Image 11 - API Keys



9. Programming Arduino

Inclusion of the necessary libraries and declaration of variables and constants inherent to the program's operation.

```
1 #include<SoftwareSerial.h>
2 #include<Wire.h>
3 #include<LiquidCrystal_I2C.h>
4 #define serialcomSpeed 115200
5 #define DEBUG true
6 #define gasPort A0
7 #define pinLedRed 5
8 #define pinLedGreen 6
9 #define pinLedBlue 7
10 SoftwareSerial ESP_Serial(10, 11); //PINOS QUE EMULAM A S
11                                     // RX_AUX --liga--> TX
12                                     // TX_AUX --liga--> RX
13 #define numeroLeds 12
14 #define anelLeds 2
15 #define tempPort A0
16 LiquidCrystal_I2C lcd(0x27, 16, 2);
17 int calculaGas(void);
18 void showTemp(int);
19 int calcgas= 0;
20 int lastGas = 0;
21 int levelGas = 0;
22 long writingTimer = 17;
23 long startTime = 0;
24 long waitTime = 0;
25 unsigned long controleTempo;
26 boolean error;
27 String APIKey = "UCIKCZ6IX1N7SRCP";
28 int calculaTemperature(void);
29 void showTemp(int);
```

Void setup() function for initializing parameters for starting the program.

```
33 void setup() {
34   Serial.begin(serialcomSpeed);
35   ESP_Serial.begin(serialcomSpeed);
36   startTime = millis();
37   InitWifiModuleESP();
38   lcd.init();
39   pinMode(gasPort, INPUT);
40   pinMode(pinLedRed, OUTPUT);
41   pinMode(pinLedGreen, OUTPUT);
42   pinMode(pinLedBlue, OUTPUT);
43   lcd.setBacklight(HIGH);
44
45 }
```



AT commands

AT commands are the basic way to configure and trigger the ESP8266 when it is under control of an external device (like an Arduino, for example).

Current AT commands are direct descendants of the so-called "Hayes Standard" from 1981, used to allow personal computers to interact with telephone connections by directly controlling a mode.

The **InitWifiModule()** function initializes the ESP8266 through AT commands.

```
46 void InitWifiModuleESP() {
47     //Este procedimento envia os COMANDOS AT para o ESP 01
48     envioDadosESP_AT("AT+RST\r\n", 2000, DEBUG); //faz reset ao modulo;
49     envioDadosESP_AT("AT+CWMODE=1\r\n", 1500, DEBUG);
50     delay(100);
51     envioDadosESP_AT("AT+CWJAP=\"Epralima|\", \"*****\" \r\n", 2000, DEBUG);
52     delay(500);
53     envioDadosESP_AT("AT+CIFSR\r\n", 1500, DEBUG);
54     delay(100);
55     envioDadosESP_AT("AT+CIPMUX=0\r\n", 1500, DEBUG);
56     delay(100);
57 }
```

The **envioDadosESP_AT(str,int,boolean)** function is responsible for sending AT commands to the ESP8266

```
59 String envioDadosESP_AT(String comando, const int timeout, boolean debug)
60 {
61     String resposta = "";
62     ESP_Serial.println(comando);
63     long int tempo = millis();
64     while((tempo+timeout) > millis()){
65         while(ESP_Serial.available()){
66             char c = ESP_Serial.read();
67             resposta+=c;
68         }
69     }
70     if(debug){
71         Serial.print(resposta);
72     }
73     return resposta;
74 }
```



The **startThingSpeakCmd(str,int,boolean)** function opens connection to ThingSpeak IoT analytics platform. The IP address of the ThingSpeak platform is: 184.106.153.149 with connection on port 80. The AT command to start ThingSpeak communication is AT+CIPSTART=PROTOCOL, IP_ADRESS, PORT.

```
106 void startThingSpeakCmd(void) {
107     ESP_Serial.flush();
108     String cmd="";
109     cmd = "AT+CIPSTART=\\"TCP\\",\\"";
110     cmd+="184.106.153.149";//Endereco IP thingerSpeak
111     cmd+="\\",80";
112     ESP_Serial.println(cmd);|
113     Serial.print("Start Commands: ");
114     Serial.println(cmd);
115     if(ESP_Serial.find("Error"))
116     {
117         Serial.println("AT+CIPSTART error");
118         return;
119     }
120 }
```

The **EscreverParaThingSpeak** function generates a string to build an API Request.

Example:

GET /update?api_key=U.....P&field1= 0&field2= 0

```
94 void EscreverParaThingSpeak(void) {
95
96     startThingSpeakCmd();
97     String getStr = "";
98     getStr = "GET /update?api_key=";
99     getStr += APIKey;
100     getStr += "&field1=";
101     getStr += String(calogas);
102     getStr += "&field2=";
103     getStr += String(levelGas);
104     getStr += "\\r\\n\\r\\n";
105     GetThingSpeakcmd(getStr);
106 }
```



The **GetThingSpeak(str)** function, is responsible for determining and sending an API Request through the AT+CIPSEND command to write to the ThingSpeak channel, returning the message received by the response from the ThingSpeak data platform. The communication will be closed if the response is not favourable.

```
122 String GetThingSpeakcmd(String getStr) {
123
124     String command="";
125     command = "AT+CIPSEND=";
126     command += String(getStr.length());
127     ESP_Serial.println(command);
128     Serial.println(command);
129     int result = ESP_Serial.find(">");
130     if(result==1)
131     {
132         Serial.print("String GET--> ");
133         Serial.println(getStr);
134         ESP_Serial.print(getStr);
135         Serial.println(getStr);
136         delay(500);
137         String messageBody = "";
138         String linha="";
139         while(ESP_Serial.available()){
140             linha = ESP_Serial.readStringUntil('\n');
141             if(linha.length() == 1){
142                 messageBody = ESP_Serial.readStringUntil('\n');
143             }
144         }
145         Serial.print("MessageBody received: ");
146         Serial.print(messageBody);
147         return messageBody;
148     }
```



The **calculaGas()** and **showGas()** These functions read the gas levels (carbon monoxide for example) and update the indicators.

```
108 int calculaGas(void){
109
110     float g=0.0;
111     int gas=0;
112     int mediagas[10];
113
114     for(int i=0; i<10; i++){
115         mediagas[i] = analogRead(gasPort);
116         gas = gas + mediagas[i];
117     }
118     gas = gas/10;
119     return gas;
120 }
```

```
121 void showGas(int gas){
122
123     if(gas<50){
124         lcd.clear();
125         lcd.setCursor(0,0);
126         lcd.print("Gas Level");
127         lcd.setCursor(0,1);
128         lcd.print(gas);
129         lcd.print(" GOOD");
130         digitalWrite(pinLedRed,LOW);
131         digitalWrite(pinLedGreen,HIGH);
132         digitalWrite(pinLedBlue,LOW);
133         levelGas=0;
134         delay(100);
135     }
136     if(gas>=50 && gas<150){
137         lcd.clear();
138         lcd.setCursor(0,0);
139         lcd.print("Gas Level");
140         lcd.setCursor(0,1);
141         lcd.print(gas);
142         lcd.print(" NOT GOOD");
143         digitalWrite(pinLedRed,LOW);
144         digitalWrite(pinLedGreen,LOW);
145         digitalWrite(pinLedBlue,HIGH);
146         levelGas=1;
147         delay(100);
148     }
```



```
149   if(gas>=150){
150       lcd.clear();
151       lcd.setCursor(0,0);
152       lcd.print("Gas Level");
153       lcd.setCursor(0,1);
154       lcd.print(gas);
155       lcd.print(" DANGER!!!");
156       digitalWrite(pinLedRed,HIGH);
157       digitalWrite(pinLedGreen,LOW);
158       digitalWrite(pinLedBlue,LOW);
159       levelGas=2;
160       delay(100);
161   }
162 }
```

Results

By analysing the graphs generated, it is possible to verify the detection of carbon monoxide (Gas) at a given moment. This can happen if a car enters an underground car park that has a lot of traffic.

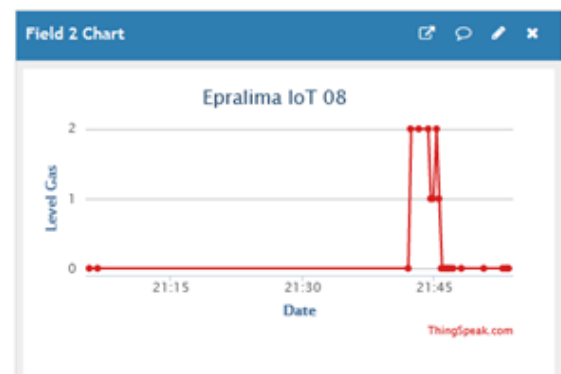
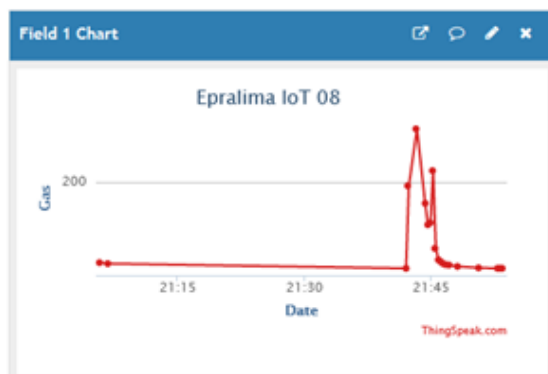


Image 12 – Results IoT ThingSpeak



The data acquired by the ThingSpeak IoT platform can also be exported to CSV files and consequently imported into datasheets as shown in Table 2

1	created_at	entry_id	field1	field2	la
2	20/04/2023 21:05	1	30	0	
3	20/04/2023 21:06	2	28	0	
4	20/04/2023 21:42	3	18	0	
5	20/04/2023 21:42	4	192	2	
6	20/04/2023 21:43	5	312	2	
7	20/04/2023 21:44	6	155	2	
8	20/04/2023 21:44	7	110	1	
9	20/04/2023 21:44	8	114	1	
10	20/04/2023 21:45	9	224	2	
11	20/04/2023 21:45	10	60	1	
12	20/04/2023 21:45	11	36	0	
13	20/04/2023 21:46	12	32	0	
14	20/04/2023 21:46	13	28	0	
15	20/04/2023 21:46	14	26	0	
16	20/04/2023 21:47	15	25	0	
17	20/04/2023 21:48	16	22	0	
18	20/04/2023 21:50	17	19	0	
19	20/04/2023 21:52	18	18	0	
20	20/04/2023 21:53	19	18	0	
21	20/04/2023 21:53	20	18	0	

Tabela 2 - DataSheet

In short

This exercise intends to apply a sensor to detect gases in the atmosphere guided by a microcontroller to alert the occupants if the surrounding atmosphere has little amount of carbon dioxide (CO₂) for example. In this way, it is possible to protect passengers from heavy atmospheres, as is the case in some car parks. This exercise is aided by a liquid crystal display (LCD) that presents written information about the amounts of CO₂.

For possible sending of data, it will be necessary to apply, for example, the ESP8266 ESP-01 module that allows the connection of several devices to the internet (or local network), and consequent sending of data from the sensors applied to the autonomous system.