Measuring pollution

Adrià Labay Mora

Francesc Pérez
IES Pompeu Fabra, Martorell
16/01/2015

Abstract

The objective of this research project is to create a device capable of analysing air pollution using an Arduino and some sensors to measure temperature, humidity, liquefied petroleum gas, ethanol, hydrocarbon gases, carbon monoxide and carbon dioxide. The measurements are sent via a Bluetooth module to an Android mobile, where they are stored and presented. It is included in the Android application capabilities like speech recognition, speech synthesis and geolocation.

Tailpipe emissions have been compared between diesel/gasoline and new/old cars.

Keywords: exhaust pipe emissions, air pollutants, Arduino, Android.

Index

1. Introduction

This project is about learning and understanding the environment, seeing what we cannot, observing what is happening around us, analysing this information and taking conclusions.

Since the natural environment is complex, this task will be focused on something that has been increasing over time: car pollution. This is causing a huge environmental problem. The levels of CO_2, O_3, SO_x… are the highest in our history, but not only is contaminating the air, it is also affecting our health.

The aim of this task is to measure the pollution emitted from different types of cars and compare them to see how they are affecting the atmosphere and also why. Understanding this brings knowledge, and with knowledge comes the power to make decisions that can change our lives for the better.

2. Materials

The measurement station is built in an Arduino One, an open-source platform based on a simple microcontroller board which makes it easy to build and develop simple circuits.

It is also required a Bluetooth module to transmit the information between the Arduino and the mobile phone. Specifically, this project makes use of a RedBearLab's Bluetooth Low Energy module. This is capable of transmitting and receiving data using this new technology which provide considerably reduced power consumption and cost while maintaining a similar communication range.

2.1. Sensors

To measure all the pollutants there are used 6 different sensors:

- **DHT11**: this sensor measures the temperature and humidity, for measuring this environment conditions it uses a resistive-type humidity measurement component and an NTC temperature measurement component. Both components are a type of resistor whose value changes significantly with humidity or temperature, respectively.
- **MQ Sensor**: this is the most common type of sensor that it is used in the project. There are 24 sensors of this type that measure different gases, although in this task there are used only 5. All this sensors use the same feature to measure the emitted gases; that is a small heater with an electro-chemical sensor.

  The electrochemical sensor operates by reacting with the gas of interest and producing an electrical signal proportional to the gas concentration. A typical electrochemical sensor consists of a hydrophobic membrane, which separates the water from the gas; a sensing electrode and a counter electrode separated by a thin layer of electrolyte (Image 1). The gas reacts at the surface of the sensing electrode involving either an oxidation or a reduction mechanism. These reactions are catalysed by the electrode materials specifically developed for the gas of interest. With a resistor connected across the electrodes, a current proportional to the gas concentration flows between the anode and the cathode. This current can be measured to determine the gas concentration.[6]

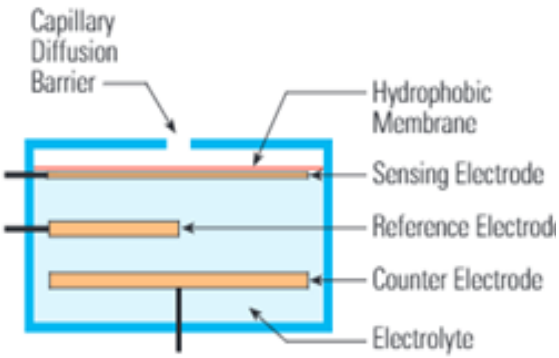


Image 1 Typical Electrochemical Sensor

A chemical reaction of this type occurs inside the sensor:

$$CO + H_2O \rightarrow CO_2 + 2H^+ + 2e^-$$

These electrons that are left in the final result, which are proportianal to the gas concentration, produce a voltage which is detected by the electrodes and transmitted to the Arduino.

In the next list are presented all the MQ sensors used in this task with the gas that they measure:

- MQ-2: measures LPG and smoke.
- MQ-3: is used to measure ethanol.
- MQ-6: is sensitive to different hydrocarbon[1] gases like methane or butane.
- MQ-7: measures the concentration of carbon monoxide (CO) in the air.
- MG-811: is sensitive to carbon dioxide (CO_2).

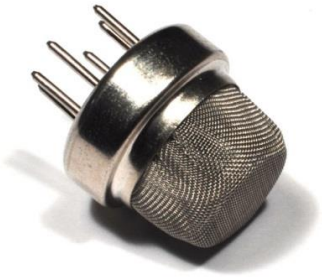


Image 2 Example of an MQ sensor

2.1.1. Problems

There was a problem with this type of sensors and the measurements. The value that receives the Arduino is a voltage between 0 and 1023 V. This value should be transformed into parts per million or µg/m^3; however, this could not be done because of the lack of information in the datasheet of this product. The reason is that, although there is a chart that relates the ppm with the sensor resistance (Rs/Ro), the latter it is not explained anywhere how to calculate it.

There was another problem with the calculation of logarithms and powers in the Arduino. These operations could give different results for the same value or even impossible results.

For these two reasons the conversion of the values is not included in the project.

[1] HC: molecules made by the combination of carbon (C) and hydrogen (H).

3. Methods

There are three separated parts in the project: the Arduino, the mobile phone and the server. These modules exchange information with each other in order to allow the process and visualization of the data. The Arduino part is the most complicated one, it is the brain: it controls the sensors and sends the measurements to the mobile. The cell phone is only used to organise and project the data, so that the users can visualize them and have access to them. Additionally, it is used to send the data to the web server. The latter, it stores these measurements whit the location from where they were acquired and finally it allows the access whenever it is needed.

3.1. Arduino

3.1.1. Circuit

In Images 3 & 4, the circuit and the schematic diagram are presented. The circuit is built in a breadboard (or protoboard) which is used to connect the different sections that form the device.

The sensors MQ3, MQ6 are connected in the same way as the MQ2, two wires are connected to the power supply; one to the ground and the other one is connected to the Arduino analogue pin through a resistor of 20kΩ. There is also the sensor MG811 which needs extra power supply to work and this is why it is connected to the Arduino and to a $9V_{DC}$ battery.

A transistor[1] is used in the connection of the MQ7 sensor. The reason behind this is that the sensor needs to be connected to 5 volts during 60 seconds and 1.4 volts during 90 seconds. In order to achieve that we use a transistor that is controlled by software. This software is included in the Arduino source code, lines 197 – 213.

The DHT sensor is connected to the power supply and the signal wire directly to the digital pin 6 of the Arduino.

The green LED is connected, through a resistor of 330Ω, to the Arduino digital pin 2 and indicates that the information from the sensors is being received.

The red LED is connected to the Arduino digital pin 13, which has a built in resistor, and is used to indicate if something is wrong.

[1] A transistor is a semiconductor device used to switch electronic signals and electrical power.

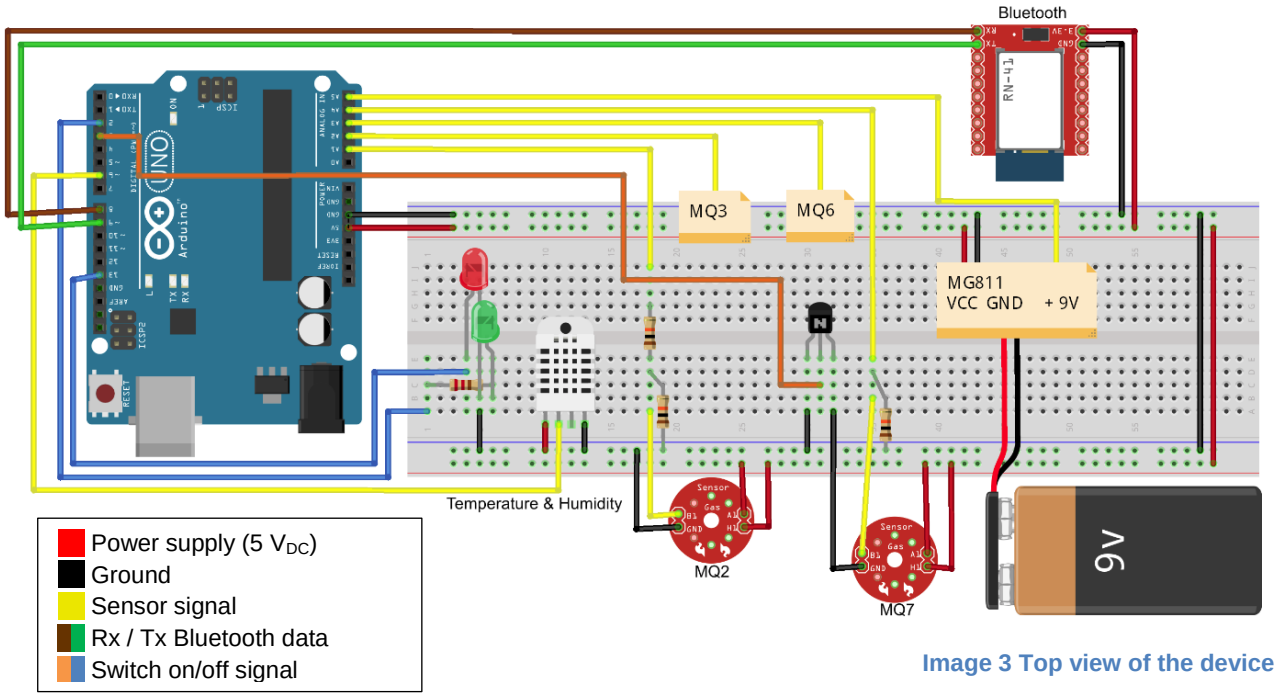


Image 3 Top view of the device

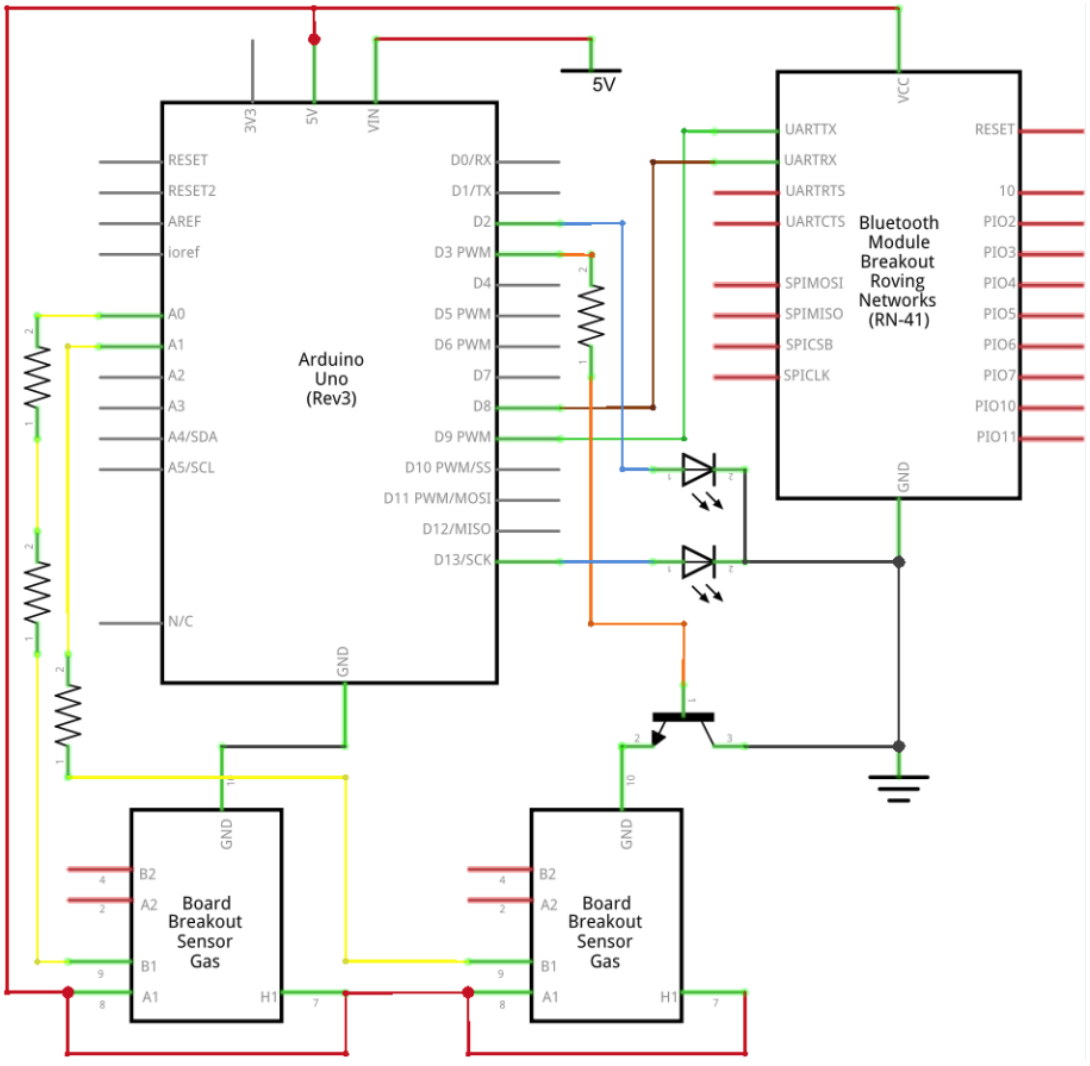


Image 4 Electric scheme

3.1.2. Software

The software allows us to control the device, ask for the measures to the different sensors and present the data to the user. There are 3 main ideas to explain:

- **Structure**: the Arduino code is a mixture of C++ and Processing (Java). There are two main functions: `setup()`, which is executed only once, when the program starts; and `loop()`, executed again and again until it is turned off.
- **Libraries**: these are extensions of the main code, completely written in C++, which are used to reduce the extension of the main code and make it simpler. 3 libraries are used: *AltSoftSerial*, that is responsible for the transmission and reception of data via Bluetooth; *DHT*, which is used for the temperature and humidity sensor, transforms the value returned by this sensor into degrees (ºC or ºF) and % of humidity; and finally *MQSensor* (Attachment 2), a library that gets the voltage measurement of the MQ Sensors and transforms it into parts per million (ppm).
- **Main code** (Attachment 1): this is the main part of the project, where in the function `setup()` (lines 41 – 74) the Bluetooth, the LEDs and the different sensors are initialized. The second part (`loop()`, lines 76 - 148) is where we get the information and send it to the mobile.

3.2. Mobile phone

This part receives the information send from the Arduino, analyses it and then saves it in an SQL[1] table containing all the pollutants values with an identifier and the date and time from when they were received. The Source code is included in Attachment 3.

This application works only on Android 4.3 or later because it is when they introduce built-in platform support for the Bluetooth Low Energy technology.

[1] SQL (Structured Query Language) is a special-purpose programming language designed for managing data held in a relational database management system.

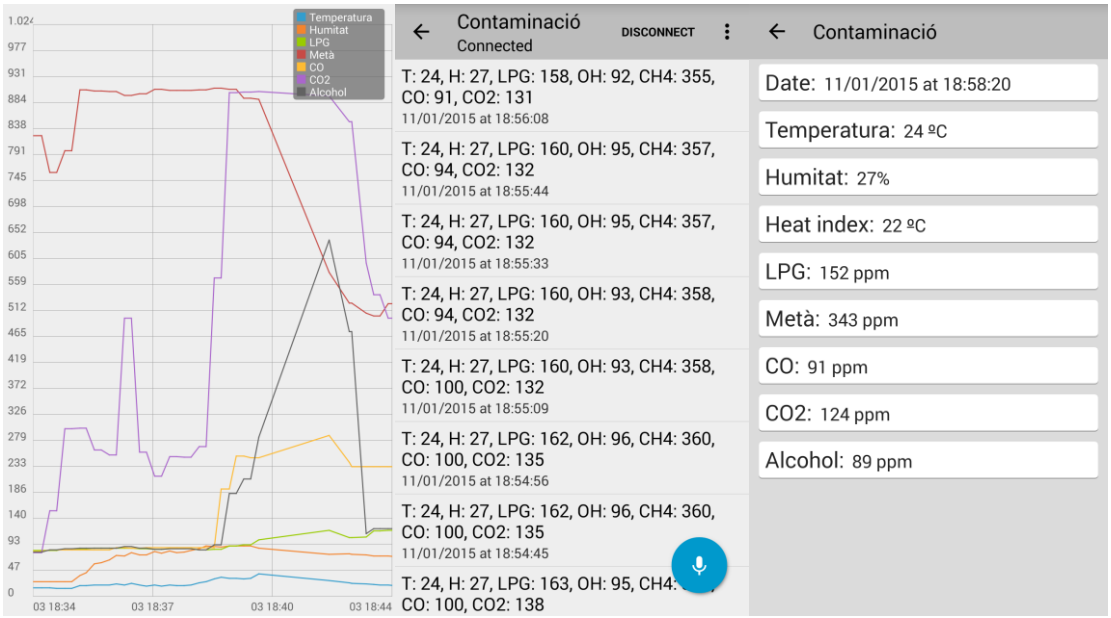


Image 5 Screenshots of the Pollution App for Android. From left to right: graph view with the data taken from the motorbike, list of the current pollution and information of all the contaminants in a specific date.

3.3. Server

A web server was created to have access to the data from any place. Visual Studio and SQL Server, both of Microsoft, were used to build it. This web server is located in the home computer and it is possible to enter by going to this URL: http://IP:2143/Api/Pollutants, where *IP* is the local router IP address. An example is shown in Image 7. The previous hyperlink will return the top 50 new pollutants, however there have been also included other methods to get specific results:

- http://IP:2143/Api/Pollutants/{city}: will return a list of the top 50 new pollutants from the city specified (if any).
- http://IP:2143/Api/Pollutants/Pollutant/{id}: will return only the pollutant that has this id (identifier).

The mobile is used to transmit the data from the Arduino to the server; as well as to see in a map the pollution in any area where there is information. Source code is included in Attachment 4.

```
SELECT TOP 10 [_id], [date], [lat], [lon], [city], [temperature], [humidity],
[lpg], [ch4], [co], [co2], [alcohol] FROM
[PollutantsDatabase].[dbo].[pollution_table] ORDER BY [date] DESC
```

	_id	date	lat	lon	city	temperature	humidity	lpg	ch4	co	co2	alcohol
1	89	2015-01-03 19:35:00	41,4737663269043	1,91301953792572	Martorell	13	81	80	432	142	915	82
2	189	2015-01-03 19:35:00	41,4737663269043	1,91301953792572	Martorell	13	81	80	432	142	915	82
3	87	2015-01-03 19:34:00	41,4737663269043	1,91301953792572	Martorell	13	79	154	429	158	915	82
4	187	2015-01-03 19:34:00	41,4737663269043	1,91301953792572	Martorell	13	79	154	429	158	915	82
5	85	2015-01-03 19:33:00	41,4737663269043	1,91301953792572	Martorell	13	76	156	423	147	915	82
6	86	2015-01-03 19:33:00	41,4737663269043	1,91301953792572	Martorell	13	77	157	428	148	915	83
7	185	2015-01-03 19:33:00	41,4737663269043	1,91301953792572	Martorell	13	76	156	423	147	915	82
8	186	2015-01-03 19:33:00	41,4737663269043	1,91301953792572	Martorell	13	77	157	428	148	915	83
9	83	2015-01-03 19:31:00	41,4737663269043	1,91301953792572	Martorell	15	76	133	376	177	915	81
10	183	2015-01-03 19:31:00	41,4737663269043	1,91301953792572	Martorell	15	76	133	376	177	915	81

Image 6 Typical SQL query. The top 10 new pollutants are selected and ordered by the date.


```
{"pollutants":[
  {"id":89,
  "date":"\/Date(1420310100000)\/",
  "city":"Martorell",
  "lat":41.4737663,
  "lon":1.91301954,
  "temperature":13,
  "humidity":81,
  "lpg":80,
  "ch4":432,
  "co":142,
  "co2":915,
  "alcohol":82},
  {...}
]}
```


Image 7 Example of the JSON returned by the following URL: http://IP:2143/Api/Pollutants/Martorell

4. Results

In this section, the results obtained with the developed system are presented. Four cars using different types of fuels will be compared. Two of them use diesel and the others use gasoline. In both cases, one of the cars is less than 3 years old and the other has 7 years or more. The petrol (gasoline) ones will also be compared to a motorbike to see the difference between these two kinds of vehicles.

In every automobile, 5 contaminants will be examined: liquid petroleum gas (LPG), different hydrocarbons (HC), carbon monoxide (CO), carbon dioxide (CO_2) and ethanol (EtOH). In order to measure these pollutants the sensors will be placed at a distance between 10 to 15 cm far from the exhaust and there will be taken 2 measurements, one at about 1000 rpm[1] and the other at 2000 rpm; the final value is the average of 10 to 15 values read from the sensor while the car was at one of this angular velocities.

For simplicity reasons, the following codes will be used to distinguish the vehicles:

- DO: diesel car older than 7 years.
- DN: diesel car younger than 3 years.
- GO: gasoline car older than 7 years.
- GN: gasoline car younger than 3 years.
- GM: gasoline motorbike older than 7 years.

4.1. Difference between diesel and gasoline

Before we start comparing the vehicles, we should first know the similarities and differences between these two types of fuels. Both combustibles come from petroleum or crude oil and are produced using fractional distillation[2] but, what makes them distinct is its calorific value: in diesel this is about 35.86 MJ/l and in petrol around 32.18 MJ/l [7]. This means that when you burn the same quantity of these fuels you get 10% more energy from diesel than from gasoline, and consequently this is the reason why in most cases a diesel car consumes less fuel than a gasoline one.

We should also mention that both fuels are principally made of hydrocarbons: the chemical formula of diesel is $C_{12}H_{23}$ and of petrol is C_8H_{18}. If we look at these two formulas we can make an assumption and say that when measuring the pollution of the cars, the value of the MQ-6 sensor measurement will be high because it measures hydrocarbons like methane and butane, the same molecules from which these two fuels are made.

[1] Revolutions per minute (rpm, rev/min or min^{-1}): is a measure of the frequency of rotation. In the SI (International System of Units) 1 rpm = 2π/60 rad/s.

[2] Fractional distillation is the separation of a mixture into its component parts by their boiling point by heating them to a temperature at which one or more fractions will vaporize.

4.2. Pollution

As it is explained above the values of these pollutants are in volts (V); however, as the values are proportional to the gas concentration, it can be seen clearly how the pollution increase or decrease in the vehicles.

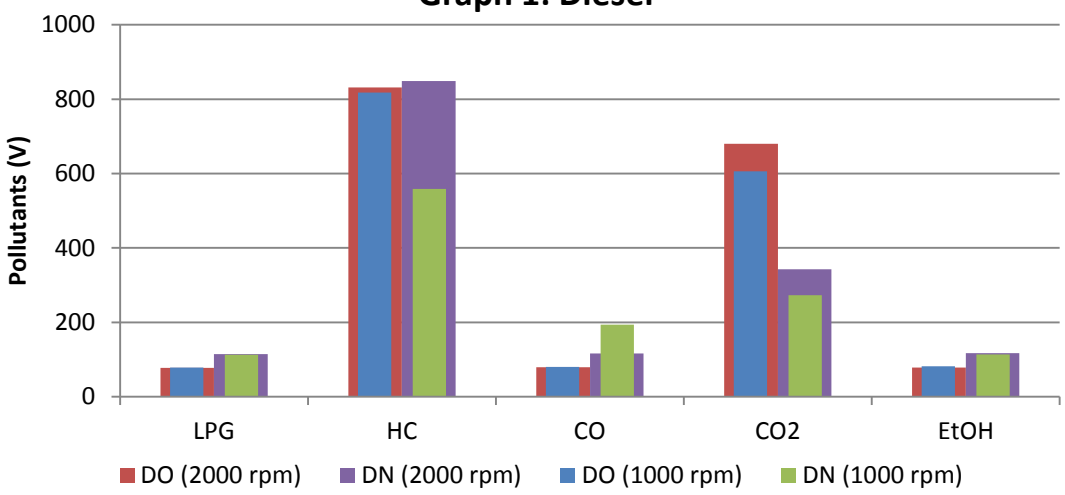
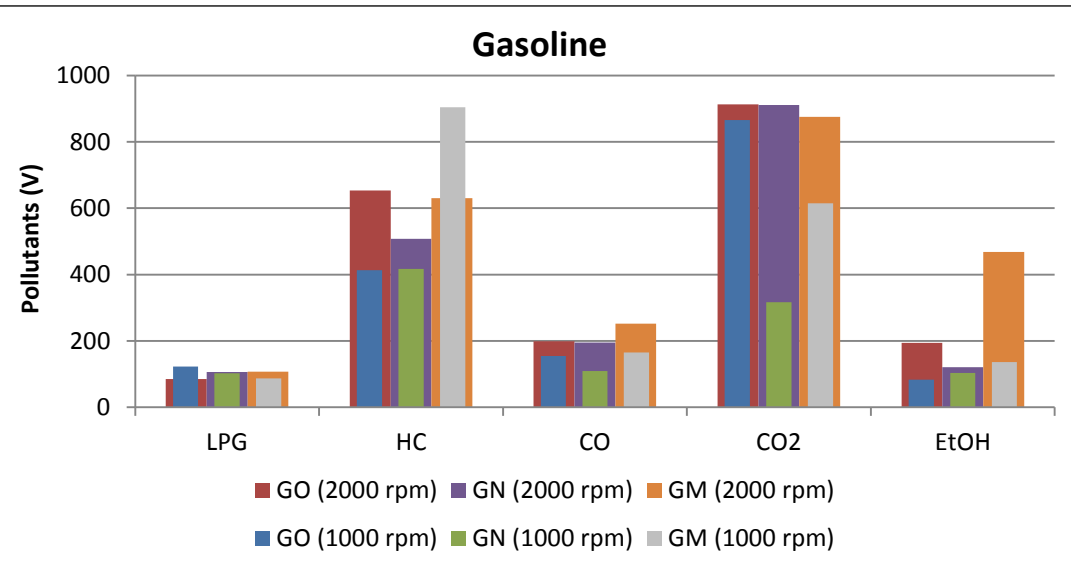


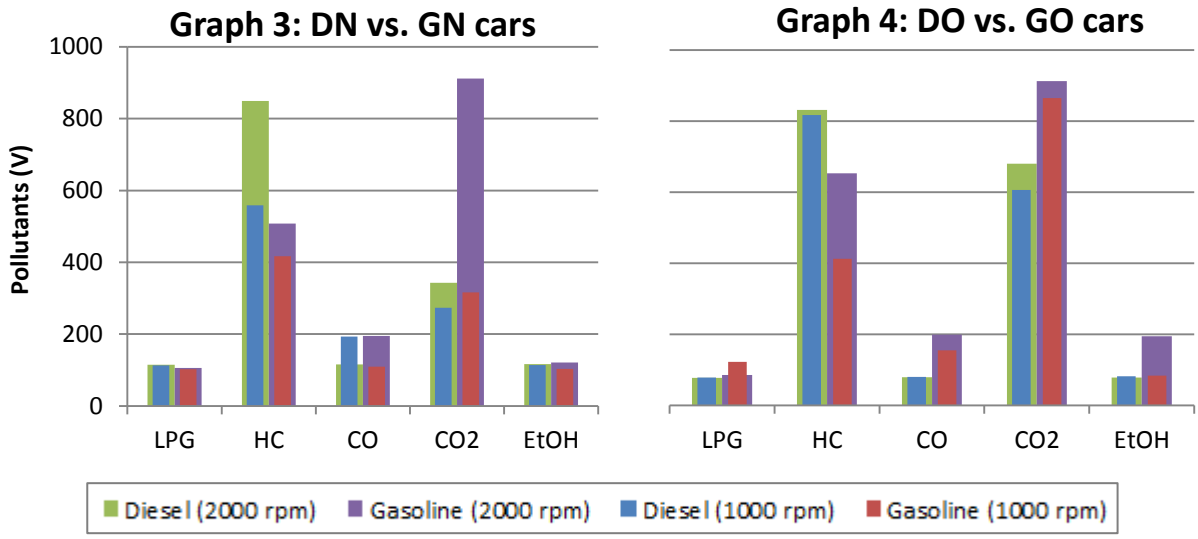
Chart that shows the different pollutants at two velocities and between two diesel cars: one old (DO) and the other new (DN).

In this first graph, it is clearly shown that two pollutants are higher than the rest, these are: HC and CO_2. The rest types of gases are always below 200 V. It is also possible to notice the difference between the old and new car; the first one emits about 300 more particles of CO_2 than the other. On the other hand, the values of HC are similar at 2000 rpm, while at 1000 rpm the emissions are lower for the new one.



Graph that shows the different pollutants at two velocities and between two gasoline cars: one old (GO) and the other new (GN); and one motorbike (GM).

The second chart is very similar to the first one if we look their structure, but the values differ a lot in some cases. First, it can be seen that the HC is lower while the CO_2 is higher in comparison with the corresponding values of Graph 1. The rest of the contaminants are all under 200 V except from the EtOH and the CO of the motorbike. Furthermore, as in the previous graph, it can be noticed that, in almost all the vehicles, the values at 2000 rpm are higher than these at 1000 rpm.



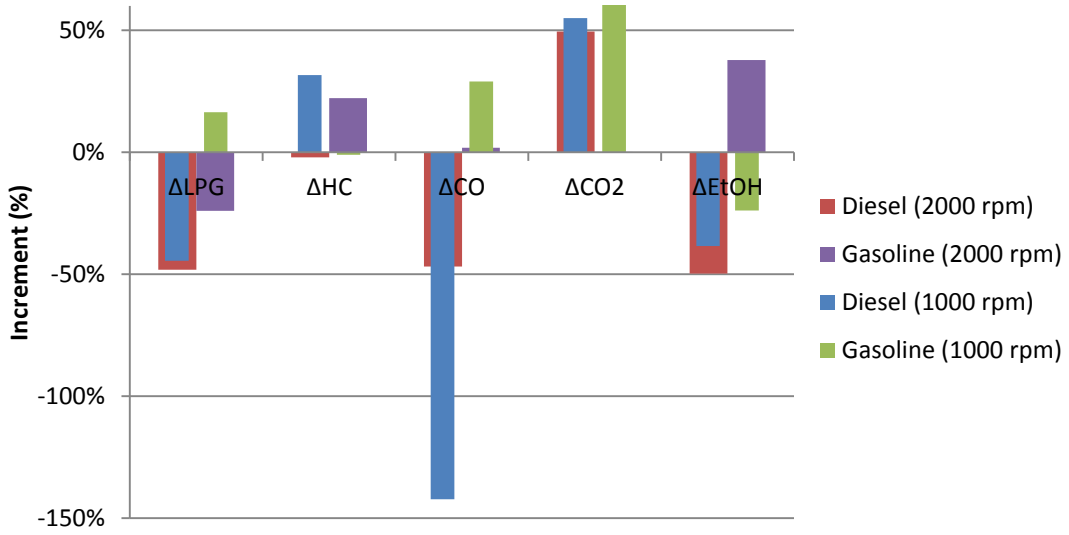
These two charts help us to see the difference between the diesel and gasoline cars, the first graph (6º) compares the new cars and the second one (7º) analyses the difference between the old cars.

From this figure, it can be clearly seen that the petrol cars are more pollutant in terms of CO_2, CO and EtOH, however the diesel cars emit more HC. In general, it can be said that the LPG's values are quite similar to each other.

4.2.1. Relations

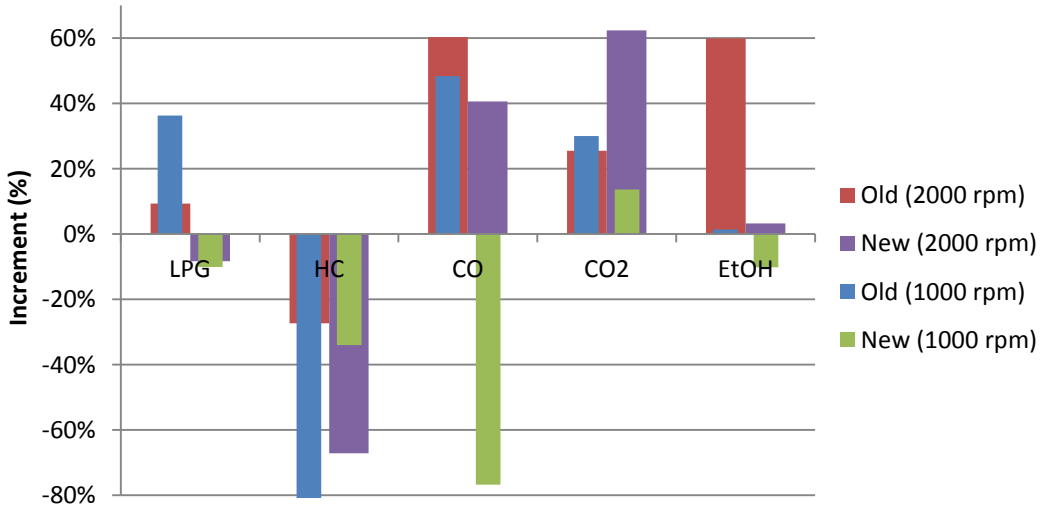
The following 3 graphs are used to show the increment of pollution between: the ages, the fuel and the rpm. The values shown are a percentage; they are only used to depict how the pollution has changed in all the cases.

Graph 5: Years (New/Old)



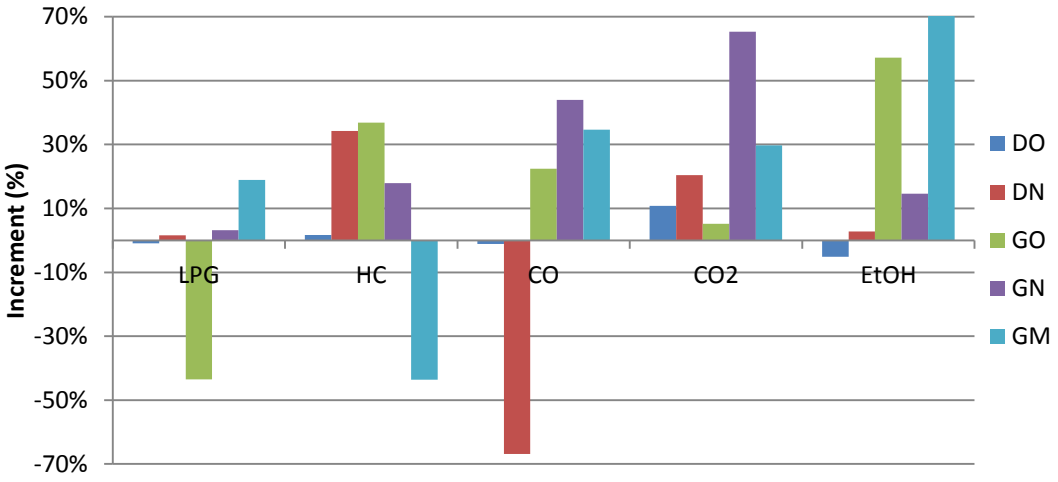
This chart shows the increment of pollution between the old and new cars. If the percentage is positive then the old car is more contaminant than the new car, if the value is lower than zero then the new car is more pollutant than the old one. Formula used: $\% = 1 - (^{New}/_{Old})$

Graph 6: Fuel (Diesel/Gasoline)



In this graph we can view the increment of pollution between the petrol and diesel cars. If the value is positive then the petrol car is more contaminant than the diesel car, if the value is lower than zero then the diesel car is more pollutant than the gasoline one. Formula used: $\% = 1 - (^{Diesel}/_{Gasoline})$

Graph 7: RPM (1000/2000)



In this diagram we can observe the increment of pollution between the different angular velocities measured. If the value is positive then the car at 2000rpm is more contaminant than at 1000rpm, if the value is lower than zero then the vehicle at 1000rpm is more pollutant than at 2000rpm. Formula used: $\% = 1 - (^{1000}/_{2000})$

5. Discussion

The first thing that is worth of discussion is why the values of HC are high in all the vehicles. This is something that was presented above in part 4.1 but now it will be further expanded. Both fuels are made, mainly, of hydrocarbons; diesel's chemical formula is $C_{12}H_{23}$ and petrol's formula is C_8H_{18}. If we take a look at these two formulas, it can be noticed that diesel has more molecules of carbon and hydrogen than gasoline. When we observe Graph 6, it can be seen that in all cases the number of hydrocarbons emitted from the diesel cars is higher because in its chemical formula there are more molecules of carbon than in the gasoline formula. We should also notice that these HCs expelled are the residues of the combustion; the more efficient a motor is the less HC molecules are emitted.

If what it has been said in the previous paragraph is true, shouldn't the diesel car expel also more CO_2 since it has more carbon molecules and consequently generate more carbon dioxide?

Diesel: $2C_{12}H_{23} + \frac{71}{2}O_2 \rightarrow 24CO_2 + 23H_2O$

Gasoline: $2C_8H_{18} + 25O_2 \rightarrow 16CO_2 + 18H_2O$

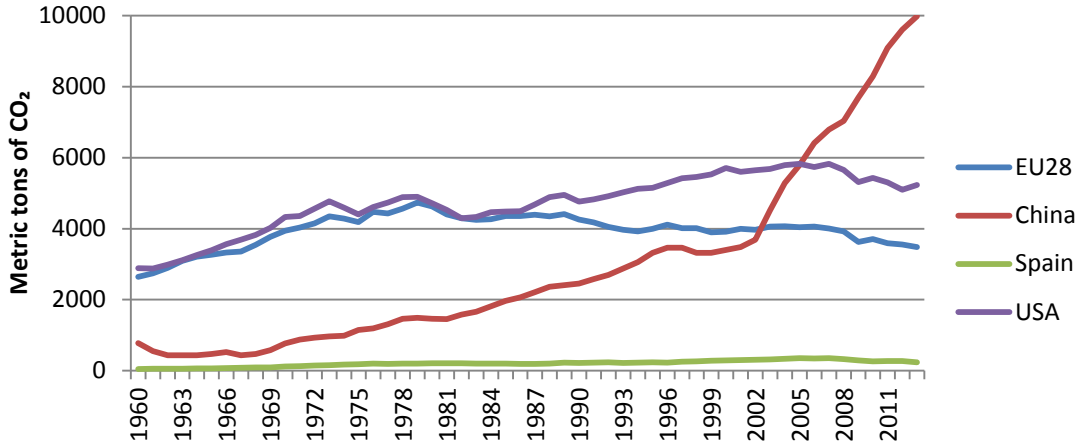
This is completely true; when diesel is burned it emits 8 more molecules of CO_2 than gasoline. So, why the measurements show that petrol emits more CO_2? It could be a measurement error but in my opinion, this error occurs because the diesel motor is more efficient and therefore it consumes less fuel. Moreover, it was tested for a short period of time and at low rpm. These factors make diesel cars less pollutant, but in real conditions it is not the same: about 2.35 kg/l of CO_2 are produced by gasoline and 2.67 kg/l of CO_2 by diesel cars [4]; probably more litres of petrol were consumed during the test and this is why the results show a less pollutant diesel car. We should also consider that, in the engine, there are other components and substances that should be included in the reaction, the result is not always CO_2 and water; there is also HC, NO_x, CO, Pb, etc.

In Graph 5 is examined how the pollution changes depending on the age of the car. It can be seen the improvement made when talking about the emissions of carbon dioxide, especially in diesel cars, whereas the emission of other gases like CO, EtOH and LPG has increased. The emission of HC has also been reduced, but in lowest quantities. We can also see how, in general, the pollution produced by gasoline cars has been reduced in bigger quantities than in the diesel ones. Whereas the expulsion of CO_2 has decreased in the new cars, worldwide it has been increasing since 1960 (Graph 8).

Graph 8 is of high interest since it shows how the levels of CO_2 emitted by fossil fuels (petrol, diesel, coal, etc.) have increased during the last decades.

Before the 21 century Europe and USA were the most contaminant states in the world with about 4000 and 5700 $MTCO_2$, respectively. In 2013, these two states had decreased both about 10% their emissions while China, nowadays the most pollutant country, has raised 65% the level of this gas. The first two have understood that pollution is a real problem and have changed their habits and started using less contaminant cars; in contrast of what it is happening in China where it seems that the environment is not one of their primary objectives.

Graph 8: Fossil fuels' emissions



This graph show how the emissions of CO_2 have increased since 1960.
Data provided by Global Carbon Atlas. URL: http://www.globalcarbonatlas.org/.

As said above, governments do care about pollution and a good example is the city of Barcelona that in December 2013 reduced the maximum velocity allowed in motorways with the target to reduce the levels of CO_2, which were over the maximum recommended. That was something controversial for most part of the population because they didn't understand why; well, the answer is in Graph 7. When we take a look at this chart, we can see that in most of the cases, as the velocity increases, the more pollutant it is. In a practical situation, when you accelerate the engine needs more fuel and as more fuel is needed, more pollutants are emitted to the atmosphere. (See below "How to reduce car pollution"). After about one week of the speed limit enforcement law, the pollution caused by the emissions of CO_2 returned to the normal levels. However, these normal levels do not imply that there is no harmful pollution, especially if compared with the atmospheric conditions of 30 years ago, before the era of the car industry revolution.

5.1. Which is more pollutant?

There are a lot of factors and conditions that make the values fluctuate, in some cases one fuel is the most contaminant whereas in others is the less.

From the mathematical point of view, diesel cars expel 12% more CO_2, 60% more HC, 10 times more NO_x, 8 times more PM and 140 times more CO; than petrol [5]. Diesel engines contribute to the problem by releasing particles directly

into the air and by emitting nitrogen oxides and sulphur oxides, which transform into "secondary" particulates[6] in the atmosphere [8]. However this does not mean that if we all had gasoline cars, there would not be any environmental problem. The Environmental Protection Agency (EPA) declared cars as "mobile sources" of pollution. Only the U.S. has 30% of the world's automobiles, yet it contributes about half of the world's emissions from cars [10].

If we see it from the practical side, this is not so easy. On the one hand we have diesel, responsible of the pollution noted above, but with two advantages: its motor and its calorific value. This motor has an efficiency about 38% while gasoline's motor (Otto cycle) is about 28% [9], apart from the fact that you obtain 10% more energy burning 1 litre of diesel that of petrol. On the other hand, we have petrol which is less pollutant, but the consumption of this fuel is higher. Overall, we cannot not know exactly which fuel is more contaminant for the atmosphere, because it can be said that there is a trade-off between the gases emitted and the level of the fuel consumption. For example, a diesel car emits 2.67 kg/l of CO_2 but it is characterized by its low consumption.

In conclusion, by evaluating the above presented results we can say that diesel is proved to be more pollutant than gasoline. This statement maybe is true but, does it really matter which is more contaminant? The real problem is the abuse of the cars by the humans. We are using the car above our possibilities, we are travelling by car to places where we could go walking or cycling. This mean of transport should only be used when there are no other options, for unexpected or long travels, not for the everyday life.

5.2. How to reduce car pollution

There is no perfect solution to avoid car pollution, but what we can do is reduce it by using some of these tips: [3]

- Avoid using cars for short journeys: walk, cycle, or take any public transport instead.
- Care for your vehicle: regular servicing helps keep your car efficient and save fuels.
- Reduce weight: too much unnecessary weight increases fuel consumption.
- Drive gently: racing starts and sudden stops increase fuel consumption. Use higher gears when traffic conditions allow it.
- Decrease the speed: at around 80 km/h emissions are very low, while they are rising dramatically above 110 km/h.
- Switch off when stationary: if stuck in traffic or stopped more than a minute. Do not leave the engine running unnecessarily.

[6] Particles created in the atmosphere by the combination of other molecules.

- Share your journeys: travel with as much people as you can, avoid using multiple cars if you can go in the same vehicle.
- Investigate alternatives: if you’re looking for a new car there are a number of different technologies and fuels available; existing cars can also be adapted to produce lower emissions.

6. Conclusion

1. A machine was built to measure and analyse the pollution emitted by cars.
2. There has been an improvement in the CO_2 emitted by the cars over the years. Although the levels of this gas continue being high.
3. The level of CO_2 emitted by fossil fuels has decreased about 10% in the USA and Europe.
4. The higher the speed, the more pollutant the car is: it does not matter the type of fuel, this is common to all the vehicles.
5. Diesel cars are generally more contaminant than petrol automobiles, but both are responsible for the high level of pollution that we actually have.
6. People, as well as the governments, are every day more aware of this problem and they try to contribute to the solution with any way they can.

The problem in fact is that people have used to travel by car everywhere because it's more comfortable, private, etc. This is the biggest problem. We need to leave this habit and start using more often public transports in our everyday life because although cars are one of the best transports that exist, they are the most polluting. In addition, another significant problem is the lack of knowledge. People are not really informed about the damage in the environment that the car abusage may cause.

With all this knowledge of what car pollution is, we can change it, we have the power and the means to improve it. It is very important to take care of our environment, to think about how it is affecting the atmosphere, what we can do and how we can change it; because what now is polluting the environment, in a near future, could be harmful to our health. We should care about what is around us because it could affect us in a future.

7. Bibliography

[1] Di Justo, Patrick; Gertz, Emily. *Atmospheric Monitoring with Arduino.* Marker Media, 2012. URL: http://shop.oreilly.com/product/0636920026686.do.

[2] Di Justo, Patrick; Gertz, Emily. *Environmental Monitoring with Arduino.* Marker Media, 2012. URL: http://shop.oreilly.com/product/0636920021582.do.

[3] *Car pollution*. Environmental protection UK, 2014. URL: http://www.environmental-protection.org.uk/committees/air-quality/air-pollution-and-transport/car-pollution/.

[4] *How much carbon dioxide is produced by burning gasoline and diesel fuel?.* U.S. Energy Information Administration. May 21, 2014 URL: http://www.eia.gov/tools/faqs/faq.cfm?id=307&t=11.

[5] Nicolás Fraile, Carlos. "Contaminación Insostenible*". DGT.es*, 2005. URL: http://www.dgt.es/revista/archivo/pdf/num170-2005-Contaminacion.pdf.

[6] *Electrochemical Sensors*. URL: http://www.intlsensor.com/pdf/electrochemical.pdf.

[7] *Diesel fuel*. URL: http://en.wikipedia.org/wiki/Diesel_fuel.

[8] *Diesel Engines and Public Health*. Union of Concerned Scientists. URL: http://www.ucsusa.org/clean_vehicles/why-clean-cars/air-pollution-and-health/trucks-buses-and-other-commercial-vehicles/diesel-engines-and-public.html#.VLAAcVVwsrg.

[9] *Diesel engines*. URL: http://en.wikipedia.org/wiki/Diesel_engine.

[10] Brinson, Linda C.. *How much air pollution comes from cars?*. HowStuffWorks.com, 29 August 2012. URL: http://auto.howstuffworks.com/air-pollution-from-cars.htm.

[11] C. Brinson, Linda. *How much air pollution comes from cars?.* How stuff works, URL: http://auto.howstuffworks.com/air-pollution-from-cars.htm

[12] Arcaklioğlu , Erol. *A diesel engine's performance and exhaust emissions*. ScienceDirect.com, January 2005. URL: http://www.sciencedirect.com/science/article/pii/S0306261904000376

[13] Orovio Astudillo *, Manuel. Contaminación en motores gasolina y diésel*. 2012. URL: http://autastec.com/blog/tecnologias-limpias/contaminacion-en-motores-gasolina-y-diesel/.

[14] E. Cano. "¿Contamina más un coche diésel o un coche de gasolina?". *ABC.es*, 2014. URL: http://www.abc.es/motor-reportajes/20140917/abci-contaminan-diesel-gasolina-201409161153.html

7.1. Other resources:

- *MQ Gas* Sensors. Arduino.cc. URL: http://playground.arduino.cc/Main/MQGasSensors.
- *MQ-2 Smoke/LPG/CO Gas Sensor Module*. February 2014, Sandbox Electronics. URL: http://sandboxelectronics.com/?p=165.
- *MQ-3 Gas Sensor Datasheet*. Hanwei Electroni Co. URL: https://www.sparkfun.com/datasheets/Sensors/MQ-3.pdf
- *MQ-5 Gas Sensor Datasheet*. Hanwei Electroni Co. URL: http://www.seeedstudio.com/depot/datasheet/MQ-5.pdf
- *MQ-6 LPG LNG Gas Sensor Module*. February 2014, Sandbox Electronics. URL: http://sandboxelectronics.com/?p=191.
- *MQ*-7 Gas Sensor Datasheet. Hanwei Electroni Co. URL: https://www.sparkfun.com/datasheets/Sensors/Biometric/MQ-7.pdf.
- *MG-811 CO2 Sensor Module*. February 2014, Sandbox Electronics. URL: http://sandboxelectronics.com/?p=147.
- Houlding, David. *CO (Carbon Monoxide) Gas Sensor Using the Arduino Uno*. March 2014. URL: http://davidhoulding.blogspot.com.es/2014/03/co-carbon-monoxide-gas-sensor-using.html.
- *MLE Mini*. RedBearLab.com. URL: http://redbearlab.com/blemini/.
- *Android Developer*. Android. URL: http://developer.android.com/index.html
- Gómez Oliver, Salvador. *Curso Programación Android*. URL: http://www.sgoliver.net/blog/?page_id=3011

8. Attachments

8.1. Arduino source code

```
#include <mqsensor.h>
#include <DHT.h>
#include <AltSoftSerial.h>

// Board          Transmit  Receive   PWM Unusable
// -----          --------  -------   ------------
// Arduino Uno        9         8         10
// Arduino Leonardo   5        13       (none)
// Arduino Mega      46        48       44, 45
AltSoftSerial ble;

// Digital pins
int voltageRegulatorPin = 3;
int ledOk = 2;
int dhtPin = 6;     // temp i hum
int ledError = 13;

// Analog pins
int mq6Pin = A1;    // mq6
int mq7Pin = A2;    // mq7
int mq3Pin = A3;    // mq3
int mg811Pin = A4; //mg811
int mq2Pin = A5;    // mq2

// MQ-7 constants
#define MQ7_HEATER_5_V_TIME_MILLIS 60000
#define MQ7_HEATER_1_4_V_TIME_MILLIS 90000

unsigned long startMillis;
unsigned long switchTimeMillis;
boolean heaterInHighPhase;

DHT dht(dhtPin);

MQSensor mq2s;
MQSensor mq3s;
MQSensor mq6s;
MQSensor mq7s;
MQSensor mg811s;

void setup() { /** Lines 41 - 74 **/
  // initialize the digital pin as an output.
  Serial.begin(9600);
  while(!Serial){}
  Serial.println("Sketch started");

  // Initialize the bluetooth
  ble.begin(57600);
  ble.println("Bluetooth started");

  // Inits the leds and turn them off
  pinMode(ledError, OUTPUT);
  pinMode(ledOk, OUTPUT);

  digitalWrite(ledError, HIGH);
  digitalWrite(ledOk, HIGH);
```

```
  // Initializes the sensors
  dht.begin();
  mq2s.begin(mq2Pin);
  mq3s.begin(mq3Pin, 200);
  mq6s.begin(mq6Pin, 20);
  mq7s.begin(mq7Pin, 10);
  mg811s.begin(mg811Pin, 1);

  pinMode(voltageRegulatorPin, OUTPUT);

  startMillis = millis();

  turnHeaterHigh();
  digitalWrite(ledError, LOW);
  digitalWrite(ledOk, HIGH);
  delay(500);
}

void loop() { /** Lines 76 - 148 **/
  // Turns the green led on.
  digitalWrite(ledOk, HIGH);

  toogleHeater();

  // Reads the sensors
  Serial.println("===========TEMP I HUM===========");
  int hum = dht.readHumidity();
  if (isnan(hum)) hum = -274;
  int temp = dht.readTemperature();
  if (isnan(temp)) temp = -274;
  Serial.print(temp);
  Serial.print(";");
  Serial.println(hum);

  Serial.println("===========MQ 2===========");
  if (mq2s.ro <= 0) mq2s.calibrate();
  Serial.print(mq2s.read(false));
  Serial.print("\t");
  int mq2 = absValue(mq2s.read(false));
  Serial.println(mq2, DEC);

  Serial.println("===========MQ 3===========");
  if (mq3s.ro <= 0) mq3s.calibrate();
  Serial.print(analogRead(mq3Pin));
  Serial.print("\t");
  int mq3 = absValue(mq3s.read(false));
  Serial.println(mq3, DEC);

  Serial.println("===========MQ 6===========");
  if (mq6s.ro <= 0) mq6s.calibrate();
  Serial.print(analogRead(mq6Pin));
  Serial.print("\t");
  Serial.print(mq6s.read(false));
  Serial.print("\t");
  int mq6 = absValue(mq6s.read(false));
  Serial.println(mq6, DEC);

  int co = 0;
  Serial.println("===========MQ 7===========");
  if (mq7s.ro <= 0) mq7s.calibrate();
```

```
  if (heaterInHighPhase) {
    Serial.print(analogRead(mq7Pin));
    Serial.print("\t");
    co = absValue(mq7s.read(false));
    Serial.println(co);
  } else {
    float last = mq7s.getLastRead();
    Serial.print(last);
    if (last > 0) {
      Serial.print("\t");
      co = last;
      Serial.println(co);
    } else {
      Serial.println("");
    }
  }

  Serial.println("===========MG 811===========");
  int mg811 = mg811s.read(false);

  Serial.println(mg811);

  String text = makeString(temp, hum, mq2, mq3, mq6, co, mg811);
  ble.println(text);

  // Turns the green led off.
  digitalWrite(ledOk, LOW);

  // Waits 10 seconds to start again
  delay(10000);
}

/**
 * Makes the string that has to be send to the mobile.
 **/
String makeString(int t, int h, int mq2, int mq3,
        int mq6, int mq7, int mg811) { /** Lines 154 - 164 **/
  int length = 2 + 6 + getIntLength(t) + getIntLength(h)
        + getIntLength(mq2) + getIntLength(mq3)
        + getIntLength(mq6) + getIntLength(mq7)
        + getIntLength(mg811);
  char c[length];
  sprintf(c, "S%d;%d;%d;%d;%d;%d;%dE", t, h, mq2,
            mq3, mq6, mq7, mg811);
  Serial.println(c);
  return c;
}

/**
 * Transforms a double value into an into,
 * making any necessary aproximation.
 **/
int absValue(double d) { /** Lines 170 - 178 **/
  int i = (int) d;
  if (i < 0) {
    i = 0;
  }

  if ((d - i) > 0.5) {
    i++;
  }
```

```
  return i;
}

/**
 * The MQ 7 sensor needs to be 90 secons recieving 1.4 volts
 * and 60 seconds recieving all the current, this is when we can read.
 **/
void toogleHeater() { /** Lines 187 - 199 **/
  if (heaterInHighPhase) {
    // 5v phase of cycle. see if need to switch low yet
    if (millis() > switchTimeMillis) {
      turnHeaterLow();
    }
  } else {
    // 1.4v phase of cycle. see if need to switch high yet
    if (millis() > switchTimeMillis) {
      turnHeaterHigh();
    }
  }
}

void turnHeaterHigh() { /** Lines 201 - 206 **/
  // 5v phase
  digitalWrite(voltageRegulatorPin, LOW);
  heaterInHighPhase = true;
  switchTimeMillis = millis() + MQ7_HEATER_5_V_TIME_MILLIS;
}

void turnHeaterLow() { /** Lines 208 - 213 **/
  // 1.4v phase
  digitalWrite(voltageRegulatorPin, HIGH);
  heaterInHighPhase = false;
  switchTimeMillis = millis() + MQ7_HEATER_1_4_V_TIME_MILLIS;
}

int getIntLength(int i) { /** Lines 215 - 229 **/
  if (i == 0) return 1;

  if (i < -999999999) return 11;
  if (i > 999999999 || i < -99999999) return 10;
  if (i > 99999999 || i < -9999999) return 9;
  if (i > 9999999 || i < -999999) return 8;
  if (i > 999999 || i < -99999) return 7;
  if (i > 99999 || i < -9999) return 6;
  if (i > 9999 || i < -999) return 5;
  if (i > 999 || i < -99) return 4;
  if (i > 99 || i < -9) return 3;
  if (i > 9 || i < 0) return 2;
  if (i > 0) return 1;
}
```

8.2. MQSensor library source code

```
#include "mqsensor.h"

#if ARDUINO >= 100
 #include "Arduino.h"
#else
 #include "WProgram.h"
#endif
```

```
MQSensor::MQSensor() {
    this->RO_CLEAN_AIR_FACTOR = 10;
}

void MQSensor::begin(int pin) {
    this->begin(pin, 20);
}

void MQSensor::begin(int pin, int rl) {
    _pin = pin;
    pinMode(_pin, INPUT);
    this->rl = rl;
    calibrate();
    lastReadTime = 0;
    lastRead = 0;
    Serial.print("Ro: ");
    Serial.println(ro);
}

/**
 * This is used to calibrate the sensor, should be executed in clean air.
 * It calculates the sensor resistance (Rs) divided by the RO_CLEAN_AIR_FACTOR.
 **/
float MQSensor::calibrate() {
    float val = 0;

    for (int i = 0;i < 5;i++) {
      val += calculateResistance(analogRead(_pin));
      delay(100);
    }

    val = val / 5;
    ro = val/RO_CLEAN_AIR_FACTOR;
    //divided by RO_CLEAN_AIR_FACTOR yields the Ro
    //according to the datasheet

    return ro;
}

/**
 * This calculates the sensor resistance. It gets the resistance of the sensor
 * 5 times and calculates the average.
 * returns the sensor resistance (Rs).
 **/
float MQSensor::read(bool res) {
    if ((lastReadTime + 15000) > millis() && lastRead > 0) {
        return lastRead;
    }

  float rs = 0;

  for (int i = 0; i < 5; i++) {
    if (res) {
        rs += calculateResistance(analogRead(_pin));
    } else {
        rs += analogRead(_pin);
    }
    delay(100);
  }
```

```
  rs = rs / 5;

  lastReadTime = millis();
  lastRead = rs;

  return rs;
}

/**
 * returns the last read value of the sensor
 **/
float MQSensor::getLastRead() {
    return lastRead;
}
/**
 * Calculates the ppm of the gas by using one point (x0, y0) of the curve
 * of the gas in the datasheet and its slope m. The y is the Rs/Ro taken
 * from the MQSensor::calculateRsRoRatio(int v) of the  value read from
 * MQSensor::read(). The formula is taken from:
```

$$\log(y) - y_0 = m * (\log(x) - x_0)$$

$$(\log(y) - y_0)/_m = log(x) - x_0$$

$$\log(x) = \frac{(\log(y) - y_0)}{m} + x_0$$

$$x = 10^{[\frac{(\log(y) - y_0)}{m} + x_0]}$$

```
**/
double MQSensor::getPpm(float x0, float y0, float m) {
    return getPpm(calculateRsRoRatio(read(true)), x0, y0, m);
}

double MQSensor::getPpm(float rs_ro_ratio, float x0, float y0,
      float m) {
    return (pow(10, (((log10(rs_ro_ratio) - y0) / m) + x0)));
}

double MQSensor::getMG811Ppm(float v, float x0, float y0, float m) {
   if ((v / DC_GAIN) >= y0) {
      return 0;
   } else {
      return pow(10, (((v / DC_GAIN) - y0) / m) + x0);
   }
}

/**
 * Divide the value given from MQSensor::read() per the ro calculated.
 *
 * returns the Rs/Ro used to calculate the ppm.
 **/
float MQSensor::calculateRsRoRatio(float v) {
    return v / ro;
}

/**
 * Calculate the sensor resistance.
 * The sensor and the load resistor form a voltage divider. Given the voltage
 * across the load resistor and its resistance, the resistance of the sensor
 * could be derived.
 * int raw_adc value read from the sensor
 *
 * returns the resistance of the sensor.
 **/
```

```
float MQSensor::calculateResistance(int raw_adc) {
    return (rl * (1023 - raw_adc) / raw_adc);
}
```

8.3. Android source code

```
/**
 * Model that contains the pollution information of one date.
 **/

public class Pollutant {

    private long id = -1;
    private Date date;
    private int temperature, humidity, lpg, smoke;
    private int buta, co, co2, alcohol, benzine;

    // Empty constructor
    public Pollutant() {}

    // Constructor with all sensor measurements
    public Pollutant(int t, int h, int mq2, int mq3,
            int mq6, int mq7, int mg811) {
        this.temperature = t;
        this.humidity = h;
        this.lpg = Math.max(mq2, 0);
        this.alcohol = Math.max(mq3, 0);
        this.buta = Math.max(mq6, 0);
        this.co = Math.max(mq7, 0);
        this.co2 = Math.max(mg811, 0);
        this.date = new Date();
    }

    // Constructor with all the pollutants.
    public Pollutant(int t, int h, int lpg, int smoke, int buta,
            int co, int co2, int al, int bz, Date date) {
        this.temperature = t;
        this.humidity = h;
        this.lpg = lpg;
        this.smoke = smoke;
        this.buta = buta;
        this.co = co;
        this.co2 = co2;
        this.alcohol = al;
        this.benzine = bz;
        this.date = date;
    }

    // Constructor for the cursor in the SQL
    public Pollutant(Cursor cursor,) {
        if (cursor == null) return;

        this.type = type;

        setId(cursor.getInt(0));
        setDate(DateUtils.getDate(DateUtils.DB_FORMAT, cursor.getString(1)));
        setTemperature(cursor.getInt(2));
        setHumidity(cursor.getInt(3));
        setLpg(cursor.getInt(4));
        setButa(cursor.getInt(6));
        setCo(cursor.getInt(7));
```

```
        setCo2(cursor.getInt(8));
        setAlcohol(cursor.getInt(9));
    }

    // Constructor for the JSONObject from the server
    public Pollutant(JSONObject o) throws JSONException {
        setId(o.getLong("id"));
        String d = o.getString("date");
        setDate(new Date(Long.valueOf(d.substring(
                  d.indexOf("(") + 1, d.indexOf(")")))));
        setTemperature(o.getInt("temperature"));
        setHumidity(o.getInt("humidity"));
        setLpg(o.getInt("lpg"));
        setButa(o.getInt("ch4"));
        setCo(o.getInt("co"));
        setCo2(o.getInt("co2"));
        setAlcohol(o.getInt("alcohol"));
    }

    /**
     * Getters and setters
     **/
    public long getId() {
        return id;
    }
    public void setId(long id) {
        this.id = id;
    }

    public int getHumidity() {
        return humidity;
    }
    public void setHumidity(int humidity) {
        this.humidity = humidity;
    }

    public int getTemperature() {
        return temperature;
    }
    public void setTemperature(int temperature) {
        this.temperature = temperature;
    }

    public int getLpg() {
        return lpg;
    }
    public void setLpg(int lpg) {
        this.lpg = lpg;
    }

    public int getButa() {
        return buta;
    }
    public void setButa(int buta) {
        this.buta = buta;
    }

    public int getCo() {
        return co;
    }
    public void setCo(int co) {
```

```
        this.co = co;
    }

    public int getCo2() {
        return co2;
    }
    public void setCo2(int co2) {
        this.co2 = co2;
    }

    public int getAlcohol() {
        return alcohol;
    }
    public void setAlcohol(int alcohol) {
        this.alcohol = alcohol;
    }

    public Date getDate() {
        return date;
    }
    public void setDate(Date date) {
        this.date = date;
    }
}

/**
 * This part of the code is taken from the Service that
 * handles the connection between the mobile and Arduino.
 * This Service is running in the background and is notified
 * when there is new data available.
 **/
public class BluetoothLeService extends Service {
    /**
     * Used to control the Bluetooth state of the device.
     **/
    private BluetoothManager bluetoothManager;
    private BluetoothAdapter bluetoothAdapter;

    /**
     * This is the Bluetooth device itself.
     **/
    private BluetoothGatt bluetoothGatt;

    /**
     * An instance of the SQL database where the
     * pollutants are saved.
     **/
    private DBHelper dbHelper;

    private String data = "";

    /**
     * Custom enum to know which is the current Bluetooth state:
     * ENABLED, DISABLED, CONNECTED, DISCONNECTED.
     **/
    public static State state = State.DISABLED;
```

```
    /**
     * A basic data element used to construct a GATT service.
     **/
    private BluetoothGattCharacteristic characteristicTx;
    private BluetoothGattCharacteristic characteristicRx;
    public final static UUID UUID_BLE_SHIELD_TX = UUID.fromString("713d0003-
503e-4c75-ba94-3148f18d941e");
    public final static UUID UUID_BLE_SHIELD_RX = UUID.fromString("713d0002-
503e-4c75-ba94-3148f18d941e");
    public      final      static      UUID      UUID_BLE_SHIELD_SERVICE      =
UUID.fromString("713d0000-503e-4c75-ba94-3148f18d941e");

    /**
     * Instance of the voice recognition class.
     **/
    private VoiceRecongnition voiceRecongnition = null;

    /**
     * Initiates the server and the objects.
     **/
    @Override
    public void onCreate(){
        bluetoothManager                    =                    (BluetoothManager)
getSystemService(Context.BLUETOOTH_SERVICE);
        bluetoothAdapter = bluetoothManager.getAdapter();
        dbHelper = new DBHelper(this);

        if (isEnabled()) state = State.ENABLED;
    }

    /**
     * Checks whether the Bluetooth is available and enable
     * in the device.
     **/
    public boolean isEnabled(){
        return bluetoothAdapter != null && bluetoothAdapter.isEnabled();
    }

    /**
     * Connects the device to the {@link mGattCallback} used
     * to retrieve the data and handle the connection state.
     **/
    public void connect(BluetoothDevice d){
        this.bluetoothGatt = d.connectGatt(this, false, mGattCallback);
    }

    /**
     * Disconnects the device if connected.
     **/
    public void disconnect(){
        if (bluetoothGatt == null) {
            return;
```

```
        }
        bluetoothGatt.disconnect();
    }

    /**
     * Stops the bluetooth connection, the recognition class
     * removes the notification.
     **/
    public void close(){
        if (bluetoothGatt == null) {
            return;
        }
        disconnect();
        bluetoothGatt.close();
        bluetoothGatt = null;
        state = State.DISCONNECTED;
        updateNotification();

        if (voiceRecongnition != null) {
            voiceRecongnition.stop();
            voiceRecongnition = null;
        }
    }

    /**
     * This is used when it connects to the Arduino so we
     * can transmit information between them.
     **/
    public    void    setCharacteristicNotification(BluetoothGattCharacteristic
characteristic, boolean enabled) {
        if (bluetoothAdapter == null || bluetoothGatt == null) {
            Log.w(TAG, "BluetoothAdapter not initialized");
            return;
        }
        bluetoothGatt.setCharacteristicNotification(characteristic, enabled);

        if (UUID_BLE_SHIELD_RX.equals(characteristic.getUuid())) {
            BluetoothGattDescriptor                descriptor                =
characteristic.getDescriptor(UUID

.fromString(RBLGattAttributes.CLIENT_CHARACTERISTIC_CONFIG));

descriptor.setValue(BluetoothGattDescriptor.ENABLE_NOTIFICATION_VALUE);
            bluetoothGatt.writeDescriptor(descriptor);
        }
    }

    /**
     * Returns a BluetoothGattService if the requested UUID is supported by the
remote device.
     **/
    public BluetoothGattService getSupportedGattService() {
        if (bluetoothGatt == null)
```

```
            return null;

        return bluetoothGatt.getService(UUID_BLE_SHIELD_SERVICE);
    }

    /**
     * Initializes the TX and RX characteristics of the gatt service
     * if it is availabe.
     **/
    private void getGattService(BluetoothGattService gattService) {
        if (gattService == null)
            return;

        characteristicTx = gattService
                .getCharacteristic(UUID_BLE_SHIELD_TX);

        characteristicRx = gattService.getCharacteristic(UUID_BLE_SHIELD_RX);
        setCharacteristicNotification(characteristicRx, true);
        readCharacteristic(characteristicRx);
        write(new byte[]{ (byte) 0xA0});
    }

    /**
     * Reads the requested characteristic from the associated remote device.
     **/
    public void readCharacteristic(BluetoothGattCharacteristic characteristic)
{
        if (bluetoothAdapter == null || bluetoothGatt == null) {
            Log.w(TAG, "BluetoothAdapter not initialized");
            return;
        }

        bluetoothGatt.readCharacteristic(characteristic);
    }

    /**
     * Read the RSSI for a connected remote device.
     **/
    public void readRssi() {
        if (bluetoothAdapter == null || bluetoothGatt == null) {
            Log.w(TAG, "BluetoothAdapter not initialized");
            return;
        }

        bluetoothGatt.readRemoteRssi();
    }

    /**
     * Writes a given characteristic and its values to the associated remote
device.
     **/
    public void writeCharacteristic(BluetoothGattCharacteristic characteristic)
{
```

```
        if (bluetoothAdapter == null || bluetoothGatt == null) {
            Log.w(TAG, "BluetoothAdapter not initialized");
            return;
        }

        Toast.makeText(getApplication(),          "Send          characteristic",
Toast.LENGTH_SHORT).show();
        bluetoothGatt.writeCharacteristic(characteristic);
    }

    /**
     * Sends a byte array to the device.
     **/
    public void write(byte[] value){
        if (characteristicTx == null) return;

        boolean b = characteristicTx.setValue(value);

        if (b) writeCharacteristic(characteristicTx);
    }

    /**
     * Sends an empty line to the device.
     **/
    public void writeln() {
        write(new byte[]{ '\n' });
    }

    /**
     * Sends a string to the device.
     **/
    public void writeText(String text){
        if (characteristicTx == null) return;

        boolean t = characteristicTx.setValue(text);
        if (!t) {
            byte b = 0x00;
            byte[] tmp = text.getBytes();
            byte[] tx = new byte[tmp.length + 1];
            tx[0] = b;
            System.arraycopy(tmp, 0, tx, 1, tmp.length + 1 - 1);

            t = characteristicTx.setValue(tx);
        }

        if (t) writeCharacteristic(characteristicTx);
    }

    /**
     * This is the callback that recieves the information from the
     * connected device.
     **/
```

```
    private final BluetoothGattCallback mGattCallback = new
BluetoothGattCallback() {
        @Override
        // Connection state
        public void onConnectionStateChange(BluetoothGatt gatt, int status, int
newState) {
            if (newState == BluetoothProfile.STATE_CONNECTED) {
                state = State.CONNECTED;
                Log.i(TAG, "Connected to GATT server.");
                Log.i(TAG, "Attempting to start service discovery:" +
bluetoothGatt.discoverServices());
                startListening();
            } else if (newState == BluetoothProfile.STATE_DISCONNECTED) {
                state = State.DISCONNECTED;
                data = "";
                close();
                Log.i(TAG, "Disconnected from GATT server.");
            }

            sendBroadcast(BROADCAST_STATE, EXTRA_STATE, state.id);
            updateNotification();
        }

        @Override
        // New services discovered
        public void onServicesDiscovered(BluetoothGatt gatt, int status) {
            if (status == BluetoothGatt.GATT_SUCCESS) {
                getGattService(gatt.getService(UUID_BLE_SHIELD_SERVICE));
                if (listener != null){
                    listener.onServicesDiscovered(gatt.getServices());
                }
            } else {
                Log.w(TAG, "onServicesDiscovered received: " + status);
            }
        }

        @Override
        // Result of a characteristic read operation
        public void onCharacteristicRead(BluetoothGatt gatt,
BluetoothGattCharacteristic characteristic, int status) {
            if (status == BluetoothGatt.GATT_SUCCESS) {
                if (UUID_BLE_SHIELD_RX.equals(characteristic.getUuid())) {
                    final byte[] rx = characteristic.getValue();
                    processDataRecieved(rx);
                }
            }
        }

        @Override
        // Result of a characteristic change operation
        public void onCharacteristicChanged(BluetoothGatt gatt,
BluetoothGattCharacteristic characteristic) {
            if (UUID_BLE_SHIELD_RX.equals(characteristic.getUuid())) {
```

```
                final byte[] rx = characteristic.getValue();
                processDataRecieved(rx);
            }
        }
    };

    /**
     * This function processes the data recieved, this data is similiar to:
     * S__:__:__:__:__:__:__:__E.
     * 'S' and 'E' indicates the start and end of the values, and them
     * are separated with a colon.
     *
     * If the data reviceved plus the previous data non processed contains
     * 'S' and 'E' then the string is pased to {@link parsePollutant()}.
     **/
    private void processDataRecieved(byte b[]) {
        if (b == null || b.length <= 0) return;

        String d = new String(b);
        Log.e(TAG, "new data: "+d);
        data += d;

        if (data.contains("E") && data.contains("S")) {
            String       p       =       data.substring(data.lastIndexOf("S"),
data.lastIndexOf("E"));
            parsePollutant(p);
            data = data.substring(data.lastIndexOf("E"));
        }

        if (listener != null){
            listener.onDataRecieved(d);
        }
    }

    /**
     * This creates a new pollutant object with the values recieved.
     * If everything is ok then the new pollutant is saved in the database.
     **/
    void parsePollutant(String d){
        if (Utils.isEmpty(d)) return;

        d = d.replace("S", "").replace("E", "").replace(" ", "");
        String s[] = d.split(";");

        Pollutant p = null;
        if (s.length == 9) {
            try {
                p = new Pollutant(Integer.valueOf(s[0]), Integer.valueOf(s[1]),
Integer.valueOf(s[2]),       Integer.valueOf(s[3]),       Integer.valueOf(s[4]),
Integer.valueOf(s[5]),
                        Integer.valueOf(s[6]),           Integer.valueOf(s[7]),
Integer.valueOf(s[8]), new Date());
            } catch (NumberFormatException n){
```

```
                n.printStackTrace();
                return null;
            }

        }

        if (p != null){
            Log.e(TAG, "new p: "+p);
            long id = dbHelper.addPollutant(p);
            p.setId(id);
            if (listener != null){
                listener.onNewPollutant(p);
            }
        }
    }

    private void changeState(State s) {
        BluetoothLeService.state = s;
        sendBroadcast(BROADCAST_STATE, EXTRA_STATE, s.id);
    }

    private void sendBroadcast(String action, String extra, int data) {
        sendBroadcast(new Intent(action).putExtra(extra, data));
    }

    /**
     * This is executed when the Service is stopped.
     * We close the connection and all the objects.
     **/
    @Override
    public void onDestroy(){
        close();
        notificationManager.cancel(NOTIFICATION_ID);
        stopForeground(true);
        isForeground = true;
        unregisterReceiver(receiver);
        bluetoothAdapter = null;
        bluetoothManager = null;
    }

    /**
     * This reciever is used to know if the Bluetooth state of the device
     * change. If it is turned off then we close the connection.
     **/
    BroadcastReceiver receiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            if (intent != null && !TextUtils.isEmpty(intent.getAction())){
                String action = intent.getAction();

                if (action.equals(BluetoothAdapter.ACTION_STATE_CHANGED)) {
                    switch  (intent.getIntExtra(BluetoothAdapter.EXTRA_STATE,  -
1)){
```

```
                        case BluetoothAdapter.STATE_OFF:
                            changeState(State.DISABLED);
                            close();
                            stopForeground(true);
                            stopSelf();
                            break;
                        case BluetoothAdapter.STATE_ON:
                            changeState(State.ENABLED);
                            break;
                    }
                }
            }
        }
    };
}

/**
 * This class is used to recognize the voice and speak in response
 * of the information recieved.
 **/
public class VoiceRecongnition implements RecognitionListener,
            Runnable, TextToSpeech.OnInitListener {

    private Context context; // Application context

    private SpeechRecognizer sr; // Used to recognize the voice

    private TextToSpeech tts; // Used to speak

    // Whether the recognizer is currently listening or not.
    private boolean isListenig = false;

    // Constructor
    public VoiceRecongnition(Context context) {
        this.context = context;
    }

    // Initiates the recognizer and the TTS.
    @Override
    public void run() {
        if (sr == null) {
            sr = SpeechRecognizer.createSpeechRecognizer(context);
            sr.setRecognitionListener(VoiceRecongnition.this);
        }

        Intent intent = new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
        intent.putExtra(RecognizerIntent.EXTRA_MAX_RESULTS, 5);
        intent.putExtra(RecognizerIntent.EXTRA_CALLING_PACKAGE,
getPackageName());
        sr.startListening(intent);

        if (tts == null) {
            tts = new TextToSpeech(context, VoiceRecongnition.this);
```

```
            tts.setLanguage(Locale.ENGLISH.ENGLISH);
        }
    }

    // Start speaking
    @Override
    public void onBeginningOfSpeech() {
        Log.e(TAG, "speechRecognizer - beggining speech");
        isListenig = true;
    }

    // Speaking stops
    @Override
    public void onEndOfSpeech() {
        Log.e(TAG, "speechRecognizer - end speech");
        isListenig = false;
    }

    // Error when speaking
    @Override
    public void onError(int error) {
        String s = "unknown";
        switch (error) {
            case SpeechRecognizer.ERROR_AUDIO:
                s = "audio error";
                break;
            case SpeechRecognizer.ERROR_CLIENT:
                s = "client error";
                break;
            case SpeechRecognizer.ERROR_INSUFFICIENT_PERMISSIONS:
                s = "insufficient permissions";
                break;
            case SpeechRecognizer.ERROR_NETWORK:
            case SpeechRecognizer.ERROR_NETWORK_TIMEOUT:
                s = "network error";
                break;
            case SpeechRecognizer.ERROR_NO_MATCH:
                s = "no match found";
                break;
            case SpeechRecognizer.ERROR_SPEECH_TIMEOUT:
                s = "speech timeout";
                break;
            case SpeechRecognizer.ERROR_RECOGNIZER_BUSY:
                s = "recognizer busy";
        }
        Log.e(TAG, "speechRecognizer - error ? " + s);
    }

    /**
     * Results of speaking. This loops throught all the results (5)
     * and analyse what they say and buy an appropiate answer.
     **/
    @Override
```

```
    public void onResults(Bundle results) {
        if (results != null) {
            ArrayList<String>                       strings                       =
results.getStringArrayList(SpeechRecognizer.RESULTS_RECOGNITION);
            if (strings != null && !strings.isEmpty()) {
                boolean b = parseResults(strings);
                if (b) {
                    message.setText(android.R.string.cancel);
                    progressBar.setVisibility(View.GONE);
                    return;
                }
            }
        }
    }

    private boolean parseResults(ArrayList<String> results) {
        for (String s : results) {
            s = removeDiacriticalMarks(s.toLowerCase());
            Log.e(TAG, "speechRecognizer - result ? " + s);
            if
(s.contains(removeDiacriticalMarks(getString(R.string.current).toLowerCase())))
{
                Cursor cursor = dbHelper.getLast(1);
                if (cursor != null && cursor.moveToFirst()) {
                    Pollutant p = new Pollutant(cursor, Pollutant.Type.ALL);
                    tts.stop();
                    tts.speak(String.format(getString(R.string.resum),
p.getTemperature(), p.getHumidity(), p.getLpg(),
                            p.getButa(),         p.getCo(),         p.getCo2(),
p.getAlcohol()), TextToSpeech.QUEUE_ADD, null);
                    return true;
                }
            }                                   else                                   if
(s.contains(removeDiacriticalMarks(getString(R.string.max).toLowerCase()))) {
                int t = 0, h = 0, l = 0, b = 0, c = 0, co = 0, al = 0;
                Cursor cursor = dbHelper.getLast(20);
                if (cursor != null && cursor.moveToFirst()) {
                    do {
                        Pollutant      p      =      new      Pollutant(cursor,
Pollutant.Type.ALL);
                        t = Math.max(p.getTemperature(), t);
                        h = Math.max(p.getHumidity(), h);
                        l = Math.max(p.getLpg(), l);
                        b = Math.max(p.getButa(), b);
                        c = Math.max(p.getCo(), c);
                        co = Math.max(p.getCo2(), co);
                        al = Math.max(p.getAlcohol(), al);
                    } while (cursor.moveToNext());

                    tts.stop();
                    tts.speak(String.format(getString(R.string.resum), t, h, l,
b, c, co, al), TextToSpeech.QUEUE_ADD, null);
                    return true;
```

```
                }
            }                                   else                                    if
(s.contains(removeDiacriticalMarks(getString(R.string.mean).toLowerCase()))) {
                int t = 0, h = 0, l = 0, b = 0, c = 0, co = 0, al = 0;
                Cursor cursor = dbHelper.getLast(20);
                if (cursor != null && cursor.moveToFirst()) {
                    do {
                        Pollutant      p      =      new      Pollutant(cursor,
Pollutant.Type.ALL);
                        t += p.getTemperature();
                        h += p.getHumidity();
                        l += p.getLpg();
                        b += p.getButa();
                        c += p.getCo();
                        co += p.getCo2();
                        al += p.getAlcohol();
                    } while (cursor.moveToNext());

                    int count = cursor.getCount();
                    t = t / count;
                    h = h / count;
                    l = l / count;
                    b = b / count;
                    c = c / count;
                    co = co / count;
                    al = al / count;

                    tts.stop();
                    tts.speak(String.format(getString(R.string.resum), t, h, l,
b, c, co, al), TextToSpeech.QUEUE_ADD, null);
                    return true;
                }
            }
        }

        tts.stop();
        tts.speak(getString(R.string.no_match),       TextToSpeech.QUEUE_FLUSH,
null);

        return false;
    }

    /**
     * We init the TTS with the current Locale and
     * if it is not available we set the default to English.
     **/
    @Override
    public void onInit(int status) {
        if (status == TextToSpeech.ERROR) {
            tts.setLanguage(Locale.ENGLISH);
        }
    }
```

```
    /**
     * Stops listening if is listening
     **/
    public void stopListening() {
        if (isListenig) {
            sr.stopListening();
        }
    }

    /**
     * Stops the voice recognitiong and the TTS.
     **/
    public void stop() {
        if (sr != null) {
            sr.stopListening();
            sr.destroy();
            sr = null;
        }

        if (tts != null) {
            tts.stop();
            tts = null;
        }
    }
}

/**
 * Class used to save and retrieve the data in an SQL database.
 **/
public class DBHelper extends SQLiteOpenHelper {
    private static final int DATABASE_VERSION = 1;
    // Database Name
    public static final String DATABASE_NAME = "pollutantsDB";
    // Table name
    public static final String POLLUTANTS_TABLE = "table_pollutants";
    // Columns names
    public static final String KEY_ID = "_id", KEY_TEMPERATURE = "temperature",
            KEY_HUMIDITY = "humidity", KEY_LPG = "lpg", KEY_BUTA = "buta",
            KEY_CO = "co", KEY_CO2 = "co2", KEY_ALCOHOL = "alcohol",
            KEY_DATE = "date";
    // SQL statement to create the table
    private static final String CREATE_POLLUTANTS_TABLE = "CREATE TABLE " +
POLLUTANTS_TABLE + " ("
            + KEY_ID + " INTEGER PRIMARY KEY AUTOINCREMENT,"
            + KEY_DATE + " DATETIME, "
            + KEY_TEMPERATURE + " INTEGER, "
            + KEY_HUMIDITY + " INTEGER, "
            + KEY_LPG + " INTEGER, "
            + KEY_BUTA + " INTEGER, "
            + KEY_CO + " INTEGER, "
            + KEY_CO2 + " INTEGER, "
            + KEY_ALCOHOL + " INTEGER, "
            + " );";
```

```
    private DBHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(CREATE_POLLUTANTS_TABLE);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        db.execSQL("DROP TABLE IF EXISTS " + CREATE_POLLUTANTS_TABLE);
        onCreate(db);
    }

    public Cursor getCursorAllPollutants(){
        return getLast(200);
    }

    /**
     * Returns a Cursor with the last pollutants.
     * @param limit of the pollutants to be returned
     * @param columns to return
     **/
    public Cursor getLast(int limit, String... projection) {
        SQLiteDatabase db = this.getReadableDatabase();
        if (db != null) {
            String columns = "";
            if (projection != null && projection.length > 0) {
                for (int i = 0; i < projection.length; i++) {
                    if ((i + 1) == projection.length) {
                        columns += projection[i];
                    } else {
                        columns += projection[i] + ", ";
                    }
                }
            } else columns = "*";
            return db.rawQuery("SELECT " + columns + " FROM (SELECT " + columns
+ " FROM " + POLLUTANTS_TABLE+ " order by " + KEY_DATE + " DESC limit
"+limit+") "
                    + POLLUTANTS_TABLE + " order by "+KEY_DATE+" ASC;", null);
        }
        return null;
    }

    /**
     * Return a single pollutant that matches the specified id.
     **/
    public Pollutant getPollutant(long id) {
        SQLiteDatabase db = getReadableDatabase();
        if (db == null) return null;
```

```
        Cursor cursor = db.query(POLLUTANTS_TABLE, null, KEY_ID + " = ?", new
String[]{String.valueOf(id)}, null, null, null);
        if (cursor != null && cursor.moveToFirst()) {
            Pollutant p = new Pollutant(cursor, Pollutant.Type.ALL);
            cursor.close();
            return p;
        }

        if (cursor != null) cursor.close();

        return null;
    }

    /**
     * Saves a new pollutant to the table.
     **/
    public long addPollutant(Pollutant pollutant){
        SQLiteDatabase db = this.getWritableDatabase();

        ContentValues values = new ContentValues();
        values.put(KEY_DATE,    DateUtils.getFormattedDate(DateUtils.DB_FORMAT,
pollutant.getDate()));
        values.put(KEY_TEMPERATURE, pollutant.getTemperature());
        values.put(KEY_HUMIDITY, pollutant.getHumidity());
        values.put(KEY_LPG, pollutant.getLpg());
        values.put(KEY_BUTA, pollutant.getButa());
        values.put(KEY_CO, pollutant.getCo());
        values.put(KEY_CO2, pollutant.getCo2());
        values.put(KEY_ALCOHOL, pollutant.getAlcohol());

        if (db != null){
            return db.insert(POLLUTANTS_TABLE, null, values);
        }

        return -1;
    }
}

/**
 * Utility class to send and get the pollutants from the web server.
 **/
public class PollutionApi {
    public static final String TAG = PollutionApi.class.getSimpleName();

    // Application context
    Context context;
    // User preferences
    SharedPreferences preferences;

    // Constructor
    public PollutionApi(Context context) {
        this.context = context;
        preferences = PreferenceManager.getDefaultSharedPreferences(context);
```

```
    }

    /**
     * Returns a list of pollutants from the specifies city.
     **/
    public List<Pollutant> getPollutants(String city) {
        String ip = getIp();
        if (Utils.isEmpty(ip)) return null;

        HttpClient httpClient = new DefaultHttpClient();

        // URL
        String url = "http://" + ip + ":2143/Api/Pollutants/" + city;
        try {
            URL url1 = new URL(url);
            url = url1.toURI().toString();
        } catch (URISyntaxException | MalformedURLException e) {
            e.printStackTrace();
        }

        HttpGet get = new HttpGet(url);
        get.setHeader("content-type", "application/json");
        get.setParams(getDefaultParams(10000));

        HttpResponse resp = null;
        try {
            resp = httpClient.execute(get);
        } catch (IOException e) {
            e.printStackTrace();
            return null;
        }

        String respStr = null;
        try {
            // Http response to string
            respStr = EntityUtils.toString(resp.getEntity());
        } catch (IOException e) {
            e.printStackTrace();
        }

        IterableJSONArray array;
        try {
            // JSONObject with the list of pollutants
            JSONObject respJSON = new JSONObject(respStr);
            // The list of pollutants
            array = new IterableJSONArray(
                    respJSON.getJSONArray("pollutants"));
        } catch (JSONException e) {
            e.printStackTrace();
            return null;
        }

        List<Pollutant> pollutants = new ArrayList<>();
```

```
        // Loops throught every JSONObject and
        // adds the pollutant to the list
        for (JSONObject o : array) {
            try {
                pollutants.add(new Pollutant(o));
            } catch (JSONException j) {
                j.printStackTrace();
            }
        }

        return pollutants;
    }

    /**
     * Sends a pollutant to the server.
     **/
    public boolean sendPollutant(Pollutant p) {
        JSONObject o = new JSONObject();
        try {
            o.put("date", DateUtils.getFormattedDate(
                    DateUtils.DB_FORMAT, p.getDate()));
            LatLng location = getLatLon();
            o.put("lat", location.latitude);
            o.put("lon", location.longitude);
            o.put("city", getCity());
            o.put("temperature", p.getTemperature());
            o.put("humidity", p.getHumidity());
            o.put("lpg", p.getLpg());
            o.put("ch4", p.getButa());
            o.put("co", p.getCo());
            o.put("co2", p.getCo2());
            o.put("alcohol", p.getAlcohol());
        } catch (JSONException e) {
            e.printStackTrace();
            return false;
        }

        HttpClient httpClient = new DefaultHttpClient();

        String ip = getIp();
        if (Utils.isEmpty(ip)) return false;

        HttpPost post = new HttpPost("http://" + ip +
":2143/Api/Pollutants/Pollutant");
        post.setHeader("content-type", "application/json");

        post.setParams(getDefaultParams(10000));

        StringEntity entity = null;
        try {
            entity = new StringEntity(o.toString());
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
```

```
            return false;
        }

        post.setEntity(entity);

        HttpResponse response = null;
        try {
            response = httpClient.execute(post);
        } catch (IOException e) {
            e.printStackTrace();
            return false;
        }

        return response.getStatusLine().getStatusCode() == 200;
    }

    /**
     * Creates the HTTP parameters with the connection timeout.
     **/
    private static HttpParams getDefaultParams(int timeout) {
        BasicHttpParams params = new BasicHttpParams();
        HttpConnectionParams.setConnectionTimeout(params,              timeout);
//connection
        return params;
    }

    /**
     * Gets the city where the Arduino is located from the
     * user preferences.
     **/
    private String getCity() {
        return preferences.getString(Utils.Keys.CITY, "");
    }

    /**
     * Gets the latitude and longitude where the Arduino is
     * located from the user preferences.
     **/
    private LatLng getLatLon() {
        return   new   LatLng(preferences.getFloat(Utils.Keys.LATITUDE,   1000),
preferences.getFloat(Utils.Keys.LONGITUDE, 1000));
    }

    /**
     * Returns the server IP address from the user preferences.
     **/
    private String getIp() {
        return preferences.getString("server_ip", "");
    }
}
```

It makes use of these third party libraries:

- Chart and Graph Library for Android by jjoe64. URL: https://github.com/jjoe64/GraphView
- Google Play Services by Google. URL: https://developer.android.com/google/play-services/index.html
- FloatingActionButton by makovkastar. URL: https://github.com/makovkastar/FloatingActionButton

8.4. Web server source code

```
namespace PollutionApplication.Areas.Api.Controllers
{
    [EnableCors(origins: "*", headers: "*", methods: "*")]
    public class PollutantsController : Controller
    {
        private DBHelper db;
        public PollutantsController()
        {
            db = new DBHelper();
        }

        // GET api/pollutants
        [HttpGet]
        public JsonResult Pollutants(string city)
        {
            if (city == null)
            {
                return Json(new { pollutants =  db.getListPollutants() },
JsonRequestBehavior.AllowGet);
            }
            else
            {
                return Json(new { pollutants = db.getListPollutants(city) },
JsonRequestBehavior.AllowGet);
            }
        }

        public JsonResult Pollutant(int? id, Pollutant item)
        {
            switch (Request.HttpMethod)
            {
                case "POST":
                    return Json(db.AddPollutant(item));
                /*case "PUT":
                    return Json(db(item));*/
                case "GET":
                    Pollutant p = db.getPollutant(id.GetValueOrDefault());
                    if (p == null)
                    {
                        return Json(new { Error = true, Message = "No pollutant
found for this id" }, JsonRequestBehavior.AllowGet);
                    }
                    else
                    {
                        return Json(new { pollutant = p },
JsonRequestBehavior.AllowGet);
                    }
                case "DELETE":
```

```
                    return Json(db.deletePollutant(id.GetValueOrDefault()));
            }

            return Json(new { Error = true, Message = "Operaci□TTP desconocida"
});
        }
    }
}

namespace PollutionApplication.Areas.Api.Models
{
    public class DBHelper
    {
        private static string cadenaConexion =
            @"Data Source=PC-ADRISQLExpress;Initial
Catalog=PollutantsDatabase;Integrated Security=True";

        private static string TABLE = @"pollution_table";

        private static string COLUMNS =
            @"_id, date, lat, lon, city, temperature, humidity, lpg, ch4, co,
co2, alcohol";

        private static string COLUMNS_NO_ID =
            @"date, lat, lon, city, temperature, humidity, lpg, ch4, co, co2,
alcohol";

        private static string DATE_FORMAT = @"YYYY-MM-DD hh:mm:ss";

        public bool AddPollutant(Pollutant p)
        {
            SqlConnection con = new SqlConnection(cadenaConexion);
            con.Open();

            string command = "INSERT INTO " + TABLE + " (" + COLUMNS_NO_ID + ")
VALUES (@a, @b, @c, @d, @e, @f, @g, @h, @i, @j, @k)";

            SqlCommand com = new SqlCommand(command, con);

            com.Parameters.Add("@a", System.Data.SqlDbType.SmallDateTime).Value =
p.date;
            com.Parameters.Add("@b", System.Data.SqlDbType.Float).Value = p.lat;
            com.Parameters.Add("@c", System.Data.SqlDbType.Float).Value = p.lon;
            com.Parameters.Add("@d", System.Data.SqlDbType.NVarChar).Value =
p.city;
            com.Parameters.Add("@e", System.Data.SqlDbType.Int).Value =
p.temperature;
            com.Parameters.Add("@f", System.Data.SqlDbType.Int).Value =
p.humidity;
            com.Parameters.Add("@g", System.Data.SqlDbType.Int).Value = p.lpg;
            com.Parameters.Add("@h", System.Data.SqlDbType.Int).Value = p.ch4;
            com.Parameters.Add("@i", System.Data.SqlDbType.Int).Value = p.co;
            com.Parameters.Add("@j", System.Data.SqlDbType.Int).Value = p.co2;
            com.Parameters.Add("@k", System.Data.SqlDbType.Int).Value =
p.alcohol;

            int res = com.ExecuteNonQuery();

            con.Close();

            return (res == 1);
```

```
        }

        public Pollutant getPollutant(int id)
        {
            SqlConnection con = new SqlConnection(cadenaConexion);

            con.Open();

            string sql = "SELECT " + COLUMNS + " FROM " + TABLE + " WHERE _id=" +
id;

            SqlCommand cmd = new SqlCommand(sql, con);

            SqlDataReader reader =
cmd.ExecuteReader(System.Data.CommandBehavior.CloseConnection);

            Pollutant p = null;

            if (reader.Read())
            {
                p = pollutantFromReader(reader);
            }

            reader.Close();

            return p;
        }

        public List<Pollutant> getListPollutants(int limit = 50)
        {
            List<Pollutant> lista = new List<Pollutant>();

            SqlConnection con = new SqlConnection(cadenaConexion);

            con.Open();

            string sql = "SELECT TOP " + limit + " " +  COLUMNS + " FROM " +
TABLE + " ORDER BY date DESC";

            SqlCommand cmd = new SqlCommand(sql, con);

            SqlDataReader reader =
cmd.ExecuteReader(System.Data.CommandBehavior.CloseConnection);

            while (reader.Read())
            {
                lista.Add(pollutantFromReader(reader));
            }

            reader.Close();

            return lista;
        }

        public List<Pollutant> getListPollutants(string city, int limit = 50)
        {
            if (city == null || city.Length == 0)
                return getListPollutants(limit);

            List<Pollutant> lista = new List<Pollutant>();
```

```
            SqlConnection con = new SqlConnection(cadenaConexion);

            con.Open();

            string sql = "SELECT TOP " + limit + " " + COLUMNS + " FROM " + TABLE
+ " WHERE city='" + city + "' COLLATE Latin1_General_CI_AS ORDER BY date DESC";

            SqlCommand cmd = new SqlCommand(sql, con);

            SqlDataReader reader =
cmd.ExecuteReader(System.Data.CommandBehavior.CloseConnection);

            while (reader.Read())
            {
                lista.Add(pollutantFromReader(reader));
            }

            reader.Close();

            return lista;
        }

        public List<Pollutant> getListPollutants(float lat, float lon, int limit
= 50)
        {
            List<Pollutant> lista = new List<Pollutant>();

            SqlConnection con = new SqlConnection(cadenaConexion);

            con.Open();

            string sql = "SELECT TOP " + limit + " " + COLUMNS + " FROM " + TABLE
+ " WHERE (lat BETWEEN " + (lat - 0.005) + " AND " + (lat + 0.005)
                + ") AND (lon BETWEEN " + (lon - 0.005) + " AND " + (lon + 0.005)
                + ") ORDER BY date DESC";

            SqlCommand cmd = new SqlCommand(sql, con);

            SqlDataReader reader =
cmd.ExecuteReader(System.Data.CommandBehavior.CloseConnection);

            while (reader.Read())
            {
                lista.Add(pollutantFromReader(reader));
            }

            reader.Close();

            return lista;
        }

        public bool deletePollutant(int id)
        {
            SqlConnection con = new SqlConnection(cadenaConexion);

            con.Open();

            string sql = "DELETE FROM " + TABLE +" WHERE _id=" + id;

            SqlCommand cmd = new SqlCommand(sql, con);
```

```
            int res = cmd.ExecuteNonQuery();

            con.Close();

            return (res == 1);
        }

        private Pollutant pollutantFromReader(SqlDataReader reader)
        {
            Pollutant p = new Pollutant();

            try
            {
                p.id = reader.GetInt32(0);
                p.date = reader.GetDateTime(1);
                p.lat = (float)reader.GetSqlDouble(2);
                p.lon = (float)reader.GetSqlDouble(3);
                p.city = reader.GetString(4);
                p.temperature = reader.GetInt32(5);
                p.humidity = reader.GetInt32(6);
                p.lpg = reader.GetInt32(7);
                p.ch4 = reader.GetInt32(8);
                p.co = reader.GetInt32(9);
                p.co2 = reader.GetInt32(10);
                p.alcohol = reader.GetInt32(11);
            }
            catch (Exception e)
            {

            }

            return p;
        }
    }
}
```

We can easily go several hours without drinking water. We can comfortably go the better part of a day without eating food. But try and go more than a few minutes without breathing.[1]