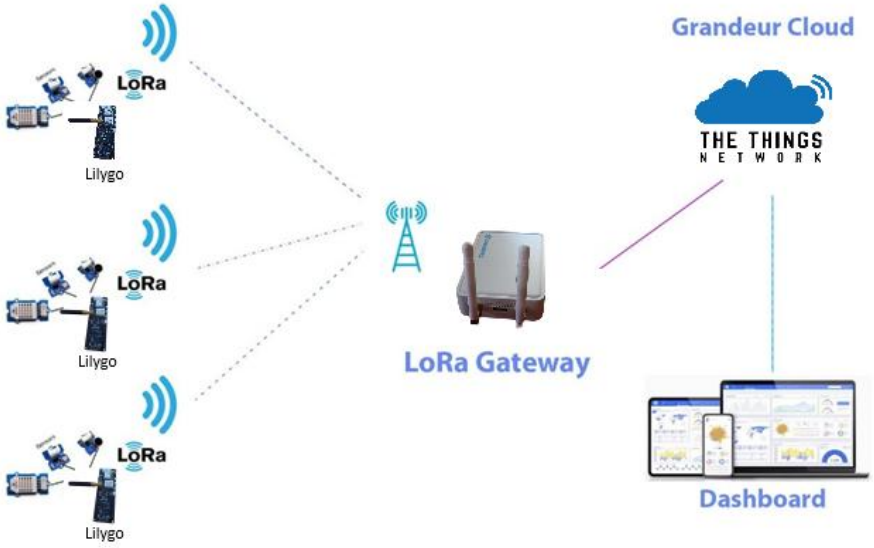

Monitoring Quality of Air in Urban Environments using LoRa Technology

BERNARDO DA SILVA BALONA

Licenciado em Engenharia Informática, Redes e Telecomunicações

Dissertação para obtenção do grau de Mestre em Engenharia de Eletrónica e Telecomunicações, no Perfil de Telecomunicações

Orientador:
Mestre Especialista Luís Miguel Rego Pires

Júri:
Presidente: Doutor Rui António Policarpo Duarte
Vogais:
Mestre Especialista Luís Miguel Rego Pires
Doutor Vítor Manuel de Oliveira Fialho

Novembro de 2025

Monitoring Quality of Air in Urban Environments using LoRa Technology

BERNARDO DA SILVA BALONA

Licenciado em Engenharia Informática, Redes e Telecomunicações

Dissertação para obtenção do grau de Mestre em Engenharia de Eletrónica e Telecomunicações, no Perfil de Telecomunicações

Orientador:
Mestre Especialista Luís Miguel Rego Pires, ISEL

Júri:
Presidente: Doutor Rui António Policarpo Duarte, ISEL
Vogais:
Mestre Especialista Luís Miguel Rego Pires, ISEL
Doutor Vítor Manuel de Oliveira Fialho, ISEL

Novembro de 2025

Agradecimentos

Em primeiro lugar, quero agradecer ao Instituto Superior de Engenharia de Lisboa por ter sido a minha casa durante estes últimos anos e me ter dado as possibilidades e ferramentas para hoje conseguir terminar esta etapa e este mestrado. Agradecer também ao professor Luís Pires por ter aceite orientar-me nesta dissertação, pela paciência e pontos de vista que foram ajudando a trilhar o meu caminho mesmo quando este se demonstrava cheio de obstáculos, foram imensos as horas a batalhar em torno dos mesmos assuntos e ao professor nunca lhe faltou a sua paciência para dar a melhor orientação possível. Um abraço e obrigado especial ao meu colega Rúben Azevedo, sem o qual, a resolução desta dissertação seria muito mais complicada.

Em segundo lugar, à minha família, ao meu pai Paulo Guilherme Bernardo Balona e à minha mãe Dora Cristina Contente da Silva Balona, que sempre estiveram do meu lado e me apoiaram mesmo quando os assuntos já ultrapassavam os seus próprios conhecimentos nunca deixaram de oferecer o ponto de vista de regressar ao ponto mais simples e voltar a trilhar pela complexidade, foi assim que resolvi alguns desafios que esta dissertação me apresentou e sem vocês a serem os meus anjos da guarda jamais seria possível.

Logo de seguida, à minha miúda e minha companheira, Jéssica Santos, pelas horas incontáveis de apoio emocional depois de longas horas voltadas nesta dissertação, pelo incansável apoio mesmo quando o fim parecia mais longe que nunca e quando as dúvidas apareceram sobre levar esta dissertação a um bom porto. Sem ti, nada disto faria sentido e sem ti, nunca chegaria onde cheguei.

Ao grande amigo que a universidade me deu e que seguiu comigo para esta aventura que foram estes anos de mestrado e licenciatura, Diogo Domingos, cuja cumplicidade demonstrada ao longo destes anos me permitiu chegar onde hoje estou. E claro sem esquecer os amigos que a licenciatura me deu, Nuno Bragança, Carolina Carreiras e Miguel Gama.

Aos amigos que a vida me deu, por todos os momentos em que me motivaram para eu poder sempre atacar esta dissertação com o máximo foco. Sem vocês para me fazerem descansar um pouco jamais teria conseguido, um grande obrigado aos meus amigos, Cláudio Pereira, Nicoleta Schitco, Inês Ribeiro, Eduardo Almeida e Joana Rebelo.

Statement of integrity

I declare that this dissertation is the result of my personal and independent research. Its content is original, and all sources listed in the bibliographic references were consulted and are duly mentioned in the text. I further declare that all scientific and technical references relevant to the development of the work are duly cited and included in the bibliographic references.

The author

Assinado por: **Bernardo da Silva Balona**
Num. de Identificação: 14372019
Data: 2025.12.15 10:16:55+00'00'

Lisbon, ……., ……

Monitoring Quality of Air in Urban Environments using LoRa Technology

Abstract

This dissertation main objective to monitor quality of air in urban environments which are densely populated using quality of air sensors, to ensure real-time monitoring, LoRa/LoRaWAN communication will be used and therefore studied in order to understand the communication in metropolitan areas.

This dissertation aims to fully experiment with the communications of values read by sensors and a LoRa cloud, all while using LoRa and a gateway. This type of project can extrapolate and evaluate the use of LoRaWAN in smart cities. By using LoRa/LoRaWAN, the objective consists of finding a low-cost option that provides real-time monitoring to measure and acquire sensor data. Moreover, this integration will allow to evaluate the viability of LoRa/LoRaWAN networks for environmental monitoring in smart cities and as a viable option for institutions to use this solution to easily monitor air quality in dense urban cities and vast countryside.

This report presents a state of art of the components the system will use such as sensors, a MCU that will employ LoRa communication and a Gateway to pass the acquired sensor data to a server cloud which will then show the values through an API, which showed expected values of concentration, percentage and other units, such as micrometers, in the atmosphere. With the dioxide carbon sensor, we could go even further in the experimentation and understood the value expected of concentration would be around 350 ppm, with the practical values being 351 ppm.

Results about values read will then be presented to understand the quality of air in the studied area and to compare these results with official values and make comparisons, measuring, the sensors' efficiency. Radio test results will also be discussed to better understand the communication length that LoRa can achieve in cities with the higher values around 1,7 km with a Spreading Factor of 7 and with a Transmitting Power of 10, 12 and 14 dBm. In addition to this entire system, it was also created a predictive system to conclude about the general quality of air with the values obtained.

Keywords: Quality of air, LoRa, LoRaWAN, Sensors, Microcontroller, IoT

Monitorização da Qualidade do Ar em Ambientes Urbanos usando Tecnologia LoRa

Resumo

Esta dissertação tem como objetivo a monitorização da qualidade do ar em ambientes urbanos densamente habitados usando sensores de qualidade do ar. De forma a assegurar o processamento em tempo-real, será usada comunicação LoRa/LoRaWAN para entender a comunicação existente nestas áreas.

Esta dissertação propõe-se a experimentar toda a comunicação que vai desde os sensores até à LoRa Cloud, tudo isto usando no meio a comunicação LoRa e uma gateway. Este tipo de projeto pode ser extrapolado para avaliar o uso de LoRaWAN em cidades inteligentes. Ao usar LoRa/LoRaWAN, o objetivo consiste em encontrar uma opção de baixo-custo que permita monitorar e obter leituras dos sensores em tempo real. Ainda mais, esta integração permitirá avaliar a viabilidade das comunicações na rede LoRa em ambiente urbanizados e também em situações de campo. Estes valores mostraram-se fieis aos valores esperados de cada parâmetro de qualidade do ar, em termos de concentração, percentagem e outras unidades, tal como micrómetros, na atmosfera. Com o sensor de dióxido de carbono pode-se ir mais além e ver uma comparação entre os valores teóricos esperados, cerca de 350 ppm, e os valores obtidos, 351 ppm.

Este relatório apresenta o estado da arte dos diversos componentes que o sistema criado usa, como sensores, microcontrolador, a comunicação LoRa e uma Gateway que depois comunica para uma LoRa Cloud através de LoRaWAN.

Os resultados obtidos são depois discutidos para compreender a qualidade do ar das áreas estudadas e realizar testes de comparação entre sensores e valores oficiais de forma a perceber a eficiência dos sensores. Resultados dos testes rádio feitos serão também discutidos para compreender as distâncias que a rede LoRa consegue alcançar nas cidades, obtendo-se um valor a rondar os 1,7 km tendo um fator de espalhamento de 7 e para as potências de emissão de 10, 12 e 14 dBm. Em adição a isto, foi ainda criado um sistema preditivo com o intuito de retirar algumas conclusões sobre os valores de qualidade do ar obtidos.

Palavras-Chave: Qualidade do Ar, LoRa, LoRaWAN, Sensors, Microcontrolador, IoT

Index

List of Acronyms

3GPP	3rd Generational Partnership Project
ABP	Activation By Personalization
ACK	Acknowledge
ADC	Analog-to-Digital Converter
AIR	Air Quality Index
AM	Amplitude Modulation
API	Application Programming Interface
AQG	Air Quality Guidelines
AQI	Air Quality Index
CDMA	Code Division Multiple Access
CSS	Chirp Spread Spectrum
DSSS	Direct Sequence Spread Spectrum
FSK	Frequency Shift-Keying
GPIO	General Purpose Input/Output
HTTPS	Hyper Text Transfer Protocol Secure
I/O	Input/Output
I2C	Inter-Integrated Circuit
IoT	Internet of Things
LoRa	Long Range
LoRaWAN	Long Range Wide Area Network
LP-WPAN	Low Rate-Wireless Personal Area Network
LPWAN	Low Power Wide Area Network
LTE	Long Term Evolution
LTE-M	Long Term Evolution for Machines
MAC	Medium Access Control
MCU	Microcontroller
MQTT	Message Queueing Telemetry Transport
NB-IoT	Narrowband-Internet of Things
OTAA	Over The Air Activation
PA	Power Amplifier
QoS	Quality of Service
RAM	Random Access Memory
REST	Representational State Transfer
ROM	Read-Only Memory
SIM	Subscriber Identity Module
SNR	Signal-to-Noise Ratio
SPI	Serial Peripheral Interface
TCP	Transmission Control Protocol
IP	Internet Protocol
TTN	TheThings Network
UART	Universal Asynchronous Receiver/Transmitter
WHO	World Health Organization
WSN	Wireless Sensor Network

List of Figures

List of Tables

1 Introduction

1.1 Context

Air Pollution is one of the greatest environmental risks to health. It was reported in 2019 that 99% of the world's population was living in places where the World Health Organization (WHO) air quality guidelines levels were not met. The combined effects of ambient air pollution and household air pollution are associated with 6.7 million premature deaths annually, out of this, 4.2 million are related to outdoor air pollution in both cities and rural areas. Exposure to particulate matter causes cardiovascular and respiratory diseases and even, cancers [1].

Most sources of outdoor air pollution are well beyond the control of individuals, and this demands a unified front between local, national and regional policies in sectors like energy, transport, waste management, urban planning and agriculture.

The WHO Global Air Quality Guidelines (AQG) offer global guidance on thresholds and limits for key air pollutants that pose health risks. The guidelines are of a high methodological quality and are developed through a transparent, evidence-based decision-making process, for example, a place that could reduce their values of particulate matter from 40 $\mu g/m^3$ to 35 $\mu g/m^3$ would save around 300 000 people from dying annually [1].

In order to study air quality, some Internet of Things (IoT) systems are being used to provide real-time monitoring, making decision making much faster. Some of them can be studied to help understand their own place in the near-future intelligent cities.

1.2 Motivation and Objectives

This dissertation focuses on the need to study and monitor air quality in real time. Quality of air is an important matter nowadays and the proximity to having someone in the family that belongs to a sensitive group makes the value of this study object much more important than ever.

Moreover, the possibility of exploring a vast and evolving area such as IoT and combining it with a problem that has been developing for more than 20 years now, shows the capacity and giant scope IoT can have.

Lastly, to study the usage of Long Range (LoRa) and Long Range Wide Area Network (LoRaWAN) in dense urban environments and evaluate its efficiency to understand its use in smart cities.

A system of a selected group of sensors was created to acquire data from particulate matter, with sensors to read gases like Oxygen, Carbon Dioxide, Carbon Monoxide and Nitrogen Dioxide, considered by WHO to be some of the most important gases to pose health risks. These sensors will then connect with a Microcontroller (MCU) which will control load balancing and sensor usage over a period of time. The results will then be sent via LoRa to a gateway which will use protocol LoRaWAN to send the values to a server cloud with Internet connection that then presents a dashboard with the values to its final user. In addition to this, a predictive system is also included to give some conclusions to values obtained from all sensor readings, in regards to labeling quality of air in measurements.

1.3 Structure

This dissertation follows a structure that resembles the way it was solved during the project's completion process. This first chapter introduces the theme to be dealt with, presenting the context on how this project came to be; it also presents the objectives this dissertation aims to achieve and the motivations behind the theme choice.

The second chapter focuses on the state of the art of all parts used to create the system for this project, such as, sensors, the microcontroller, the gateway and the LoRa Cloud, each of these with its own sub-chapter.

The third chapter is all about the LoRa network, explaining chirps and modulation processes. Information about the structure of a LoRa packet and how payload works is also present in this chapter. Last two subsections study the differences between two different ways to access the network and the choosing process of the Spreading Factor (SF) and transmitter power.

Fourth section is centered on presenting the system architecture. The chosen components and the way the system works are also present in this chapter. Sub-chapters focusing on mathematical equations used by the sensors and the creation of a predictive system also enter this section.

Fifth section presents the configuration of both the Gateway and the LoRa Cloud to work with the microcontroller. And then all the radio tests and sensor tests, finishing with a small experience which evaluates sensors performance in a smoking environment.

Lastly, a sixth section focusing on gathering all the conclusions taken over this project's timespan and presenting ways this project could move forward in the bibliography. Finishing this dissertation are three annexes referring to the configurations of both Gateway and LoRa Cloud and with the source code used in both Nodes of the MCU.

2 State of the Art and Related Works

This chapter aims to present the state of the art about LoRa technology and LoRaWAN, sensors and MCUs that are used to measure quality of air. To highlight each one of these points, section 2.1 is divided in various sub-sections dedicated to exploring fundamental aspects of each topic. Section 2.2 will be dedicated to related works that have been done in the area.

2.1 State of the Art

LoRa technology is limited in densely populated areas, where interference might suppress communication's reliability, and its range of 15 km is much lower due to the presence of buildings, in water the range can go up to 30 km [2].

LoRaWAN has been projected for low data rates, in networks with lots of devices, mainly sensors that will be used and send data with their values, traffic might cause collisions and worse, losses [3].

Nowadays, development in Low Power Wide Area Networks (LPWANs) has allowed for a wide range in innovations in IoT technologies. Among those, LoRa modulation stands out for its balance between geographical coverage, low consumption and low cost, being widely used in remote monitoring applications, smart cities, precision agriculture and industrial control. LoRaWAN protocol complements this technology by providing an efficient and scalable infrastructure for communication between devices and gateways that are geographically scattered [4].

A LPWAN connects low-bandwidth battery-powered with low bit rate over a long range, or in other words, it features wide coverage, lower power consumption and large capacity. LPWAN technologies are characterized by long range links, in the order of kilometers and usually have star network topologies [5]. It works between 868 and 921 MHz band.

Similar with cellular networks, and by typically implementing a star topology, LPWAN manages to have each node communicating with the base station directly. Unlike other Wireless Sensor Network (WSN), which have mesh and tree topologies putting the complexity in the end devices, LPWAN allocates the complexity to the base station while keeping end devices relatively simple and thus low-cost and low consumption. Therefore, it has been able to span over a variety of competing technologies, such as Sigfox [6], LoRa [7], 3rd Generation Partnership Project (3GPP) [8] originating Long Term Evolution for Machines (LTE-M), a type of 4G cellular network specifically designed for the IoT [9], and Narrowband Internet of Things (NB-IoT) [10], Weightless [11], among others [12]. One of these technologies is developing quickly in the market is LoRa [13].

When compared to others that also are in use in the field of monitorization such as Sigfox, a network operator for LPWAN connectivity product, Sigfox is operated as a public network with coverage in a subset of countries and in urban areas has a range of 10 km which is itself a limitation compared to LoRa's 15 km. Sigfox also has a limited bandwidth and message size (12 bytes per message) [6].

NB-IoT has a simpler waveform, the technology consumes minimal power and is economically friendly due to having chipsets specifically engineered for some of its protocols which in turn guarantee a simpler construction. NB-IoT also has potential for smart cities applications. However, it requires cellular network infrastructure and Subscriber Identity Module (SIM) cards and ends up having higher latency [10].

Another good example is Zigbee [14], a wireless protocol widely used in IoT devices, which offers versatility and compatibility. Zigbee has good mesh networking capability and low power consumption but is, however, restricted to a very short range (10 to 100 meters) and requires a gateway for Internet connectivity, only being a viable solution for indoor air monitoring within buildings [14].

2.1.1 Long Range Wide Area Network (LoRaWAN)

LoRaWAN technology is an open global standard provided by the association of companies known as LoRa Alliance™ [15]. LoRaWAN is based on the LoRa modulation, a proprietary spread spectrum modulation scheme, that is based on Chirp Spread Spectrum (CSS). The technology trades in data rates (250 bps to 50 kbps) for sensitivity within a fixed channel bandwidth (developers of this system might select from different spreading factors, which allows to trade in data rate for range or power to optimize the network performance and its parameters. In the context of an ISO/OSI model, LoRaWAN would be considered a MAC layer and the LoRa modulation the physical layer. The LoRaWAN technology operates in Europe within the 868 MHz band (a 250 kHz will be used for channel width) using three default channels, 868.1, 868.3 and 868.5 (all in MHz). Uses CSS which is used to detect incoming messages and allows to communicate below the noise level. These parameters are then able to provide long communication range (about 5 km in cities) without any requirements for high transmitting power. Multiple access is based in Pure ALOHA with unscheduled uplinks, however collisions are possible like this. ALOHA can also be implemented with duty-cycle where each node transmits independently, with no coordination.

LoRaWAN can be divided into three different classes, (i) Class A-end-devices with bidirectional communication (each uplink being followed by two downlinks), (ii) Class B, which is the same as Class A, additionally there are strictly defined windows, and (iii) Class C, continuously open downlink windows which are closed only during uplink time [16].

LoRaWAN allows flexibility in the use of gateways, all while regulating the duty cycle to 1% or 0.1% and the power output to 20 mW [17]. Gateways can be used to receive IoT data and alternatively, the microcontroller can connect to a commercial network of gateways [18].

One of LoRaWAN main factors and that will be seriously taken into consideration is its payload, therefore, not all sensor values will be sent through LoRa communication at the same time and will have different time periods to send information.

LoRaWAN operates on the Data Link Layer and is responsible for the encapsulation to Network Layer, this meaning that this technology specifies the Medium Access Control (MAC) and data multiplexing at the second layer, organizing communication between the end devices and gateways. It is also responsible for forwarding data between end devices, gateways and network servers. The encapsulation in IP datagrams occurs in the gateways, where packages and protocols, such as Hypertext Transfer Protocol (HTTP) (belonging to layer 5) which will be used later to send data to the dashboard, are encapsulated in transport protocols, like Transmission Control Protocol (TCP) to be sent to the cloud.

End-devices can be activated in the LoRaWAN network in two different methods, (i) Over The Air Activation (OTAA) and (ii) Activation by Personalization (ABP). (i) OTAA is considered a much safer and more flexible way to activate devices in the network. With OTAA, devices have a negotiation process with a server to get session keys which are necessary for communication. This works by having packets join request, containing the Device Extended Unique Identifier (DevEUI), the Application Extended Unique Identifier (AppEUI) and the Device Nonce (DevNonce) which is a random number, sent to the server which responds after with a join accept which contains the Application Nonce (AppNonce), the network's parameters and encrypted session keys. With all this information, two keys are generated, the Network Session Key (NwkSKey) and the Application Session Key (AppSKey). OTAA is considered more dynamic, allows for mobility, as it allows handovers, but also introduces initial overhead and mandatory connection with a server.

In (ii) ABP, devices are pre-configured manually with session keys and other network parameters, no dynamic activation process is needed. The device starts transmitting as soon as it is connected. There is no initial overhead, and the device becomes server independent, however, there is less security, no key renovation and every parameter change needs to be manually done. Handovers are also not allowed, which means the device needs to stay stationary.

Gateways receive data and send it directly to the cloud. The gateway localization should be strategic to guarantee the best coverage.

The LoRa cloud will receive the encapsulated packages from the gateway, decodes them and authenticates the data that will then be sent to the server. The user

can then make a Message Queueing Telemetry Transport (MQTT) request or find information on the API that the cloud offers if data needs to flow continuously, this also allows for a efficient bidirectional communication. When the user only needs to receive some data or integrate web services which do not support MQTT, then HTTP can be used. Therefore, a LoRa cloud can be considered as a bridge between the end devices and network servers/dashboard where its final user can access the collected data.

2.1.2 Air, Dust, Temperature and Humidity Sensors

To measure quality of air, different types of sensors were be used. They specialize in distinct areas such as gas, dust, humidity and temperature. It needs to be considered that sensors need to be easily accessible and have good efficiency in their area of expertise. Most sensors use a power supply of about 5 V but some might use 3.3 V, moreover, LoRa/LoRaWAN only supports communication with 3.3 V. Therefore, these were changes the microcontroller also needed to implement. Joining said sensors with LoRa networks will allow for real-time monitoring with a wide range of variables, such as temperature, humidity, pressure and air quality, making said networks robust to environmental scenarios and load balancing.

Important sensors for measuring quality of air are separated in three different types, (i) gases, (ii) dust, and lastly (iii) others.

Regarding group (i), gases like Oxygen (O_2) (values should be around 20-25% O_2) [19], Carbon Monoxide (CO) (values should be lower than 10 mg/m^3 in average 8-hour periods) [19], Carbon Dioxide (CO_2) (expected values should be around 0.035% CO_2) [19], Nitrogen Dioxide (NO_2) (values are not supposed to be higher than 200 μm/m^3 for periods longer than 1 hour or higher than 40 μm/m^3 in a yearly measure) [19]. These sensors are regarded as very important such as our air composition in the atmosphere.

Oxygen values should remain at those levels as the higher concentration can also be bad for human health, causing damage to lungs and restraining the gas exchange between blood and cells, this being maintained by the higher presence of Nitrogen [20]. Carbon Dioxide should also maintain said values due to the fact it is toxic to humans.

Carbon Monoxide should also be included in this study because, if found in higher concentrations, it can lead chest pain in people with heart disease, fatigue, impaired visions, nausea, headaches, among others. This odorless gas can be found in leaking chimneys and furnaces, gas stoves, generators and other gasoline-powered equipment and tobacco smoke [21].

Nitrogen Dioxide is very important in regard of being present in emissions from cars, trucks and buses [22]. If found in higher concentrations, it can irritate airways in human respiratory system, and aggravate respiratory diseases, particularly asthma. If

prolonged, humans can develop asthma and potentially increase susceptibility to respiratory infections [22].

For group (ii), dust sensors need to be particularly strong in measuring PM2.5 and PM10, as, if in high levels, are bad for human health [23]. Values differ for both types of particles, for PM2.5, yearly values should be lower than 20 $\mu g/m^3$, whereas for PM10, yearly values should be lower than 40 $\mu g/m^3$ and hourly values lower than 50 $\mu g/m^3$ [19].

Airborne Particulate Matter is not a single pollutant, but rather a mixture of many chemical species. It is a mixture of solids and aerosols composed of small droplets of liquid, dry solid fragments and solid cores. Particles vary widely in size, shape and even chemical composition. The main difference between PM2.5 and PM10 is the size, with the first referring to all particles that are 2.5 microns or less in diameter and the later referring to all particles with 10 microns or less in diameter. With this size, it becomes important to monitor their quantity in the atmosphere, because PM2.5 might be inhaled and be deposited in deeper parts of the lung and PM10 can stay in larger areas. This can cause premature mortality, heart lung diseases, chronic bronchitis, asthma attacks, among others [23].

Lastly, in group (iii) humidity and temperature sensors and even a Global Positioning System (GPS) receiver or antenna which the microcontroller can already have embedded, for location of said quality of air tests.

The second important thing needing to be taken into consideration is the output type of each sensor, which can be the Universal Asynchronous Receiver/Transmitter (UART), Analog-to-Digital Converter (ADC) or even an Inter-Integrated Circuit (I^2C) or Serial Peripheral Interface (SPI).

Table 2-1 shows all the different sensors for air quality measurements that were talked above.

Table 2-1 - Different types of sensors for air quality measurement

Group	Pollutant / Parameter	Typical Values	Importance
Gases	O_2	20 – 25%	High concentration may cause lung damage
	CO	< 10 mg/m^3 (8 hour-period)	Toxic, causes ches pain, nausea and headache
	CO_2	~0.035%	High values are toxic and a health hazard
	NO_2	< 200 μg/m^3 (1h); < 40 μg/m^3 (annual)	Irritates respiratory airways, asthma, higher risk of infection
Particulate Matter (Dust)	PM2.5	< 20 μg/m^3 (annual)	Stay deep in the lungs; asthma; higher mortality rate
	PM10	< 40 μg/m^3 (annual); < 50 μg/m^3 (day)	Superior airways restricted, cough, asthma, bronchitis
Others	Humidity	-	Necessary for environmental correlations
	Temperature	-	Necessary for sensor compensation

2.1.3 Microcontroller with LoRa and LoRaWAN Support

MCUs are a compact integrated circuit designed to govern a specific operation in an embedded system. A typical MCU includes a processor, memory and Input/Output (I/O) peripherals on a single chip. Simply place, a microcontroller is a simple miniature

PC designed to control small features of a larger component without a complex front-end operating system [24].

A MCU is embedded inside of a system to control a single function in a device. It uses its central processor to interpret data it receives from it's I/O peripherals. In this project's environment, the microcontroller will be responsible for controlling LoRa communications, load balancing and the data gathered by sensors in a *site* that will be sent to a LoRa cloud through a gateway.

The CPU processes and responds to various instructions that direct the MCU function. This involves performing basic arithmetic, logic and I/O operations. It also performs data transfer operations, which can communicate commands to other components I the larger embedded system.

The MCU's memory stores the data the processor receives and uses to respond to instructions. It's programmed to carry out. Load balancing and efficiency when using sensors will be possible due to this part of the MCU.

There are two main memory types. Program memory, which stores long-term information about instructions that the CPU carries out, this memory is non-volatile, which means it stores information over time without needing a power supply. Data Memory, this temporary data storage is used while instructions are being executed. It is volatile and therefore, it is temporary and is only maintained if the device is connected to a power source. The two main types can then branch into others, like Random Access Memory (RAM) which is used to store temporary data while the MCU is in execution. It includes variables, a stack and intermediate data. RAM is volatile data. On the other hand, non-volatile data, Read-Only Memory (ROM), which stores permanently the code or data which won't likely change. In modern MCUs, it is frequently substituted or represented by FLASH memory. Lastly, FLASH memory, which stores the firmware and, in some cases, user's configurable data, this type of memory is also non-volatile.

These are the interface for the processor to communicate with the outside world. The input ports receive information and send it to the processor in the form of binary data. The processor then receives that data and sends the necessary instructions to output devices which execute tasks external to the MCU.

There are different types of MCUs that work well or integrate LoRa technology like STMicroeletronics (STM) STM32 [25], Nordic Semiconductors nRF52840 [26], which are commonly used with LoRa due to their equilibrium between efficiency and low power consumption. STM32WL [27] which combines an ARM Cortex-M4 Core with a LoRa transceiver, and finally, ESP32 [28] LoRa which is a versatile MCU that supports Wi-Fi and LoRa.

STM32 [25] is a module that can be connected with LoRa modules like SX1276 [29] using a Serial Peripheral Interface (SPI) [29] for communication and use additional pins for general-purpose input/output (GPIO) [30] for additional resources such as

interruptions, STM uses firmware libraries such as LoRaMac-node or RadioHead [31] for managing LoRa communication. STM also has several disadvantages and limitations such as no connection with Wi-Fi or Bluetooth, complexity and high price which can be tackled with having cheaper MCUs that can also work with LoRa modules.

NRF52480 [26] can also relate to a LoRa module via SPI and manage LoRa communication. Typically connecting LoRa devices to Bluetooth networks and for sensors who use LoRa for communication and Bluetooth for local configuration. This MCU can use a software library like Adafruit Feather [32] which is compatible with platform Arduino IDE [33] and CircuitPython [34] to connect with LoRa's external modules. On the other hand, NRF52480 has some limitations like no native LoRa and different operation frequency because it was projected for 2.4 GHz differing from the frequencies LoRa operates.

STM32WL [27] is a MCU that combines an ARM Cortex-M4, used for principal tasks and data processing, with a LoRa transceiver, eliminating the need for external modules for LoRa communication. It supports LoRa and has LoRaWAN compatibility. It works with LoRa frequency and has low-power consumption. STM32WL has configurable interfaces such as SPI, I2C, UART, ADC and GPIO. It is however limited by its complexity and initial cost.

ESP32 [28] combines Wi-Fi with LoRa. It integrates GPIO, SPI, UART, I2C and ADC interfaces with 16 ADC channels. It has low-power consumption, using Light or Deep-Sleep modes, where it shuts off certain devices and functions, especially needed when using batteries. Economically accessible intertwined with great efficiency and supports different software languages such as C, C++, Python, among others. It has, however, a complex configuration at the beginning, more than others and is sensible to batteries.

Other MCUs like Arduino UNO [35], have no LoRa module but possess different interfaces (UART, I2C, SPI, GPIO and ADC) that might relate to external LoRa modules. Arduino UNO is widely used due to its simplicity and highly user-friendly software interface Arduino IDE, with lots of libraries ready to be used. Resistant to connection error and a cheap option compared to other MCUs. Arduino UNO is limited by an 8-bit processor, small memory and no native connection to Wi-Fi or Ethernet. When choosing an Arduino-UNO, sensors that operate with a 3.3 V should also not be chosen because this MCU is not adapted in a logical level to this voltage.

A LoRa node is composed of an MCU, sensors and a LoRa communication module, it is very important to also know that an IoT system needs to have a low-power consumption so that the battery can last for longer. The sensors current consumption must, therefore, be taken into consideration. The battery can be recharged by using solar panels or using other harvesting systems that use vibration or temperature, among others.

Table 2-2 sums all the information by presenting a comparison between different MCUs that can support LoRa.

Table 2-2 - Comparison between stated MCUs

MCU	Integrates LoRa?	Interfaces	Advantages	Disadvantages
STM32	No (uses an external SX1276)	SPI, GPIO	Firmware Libraries such as LoRaMac-node, RadioHead	No Wi-Fi or Bluetooth; Complex; Cost
NRF52840	No (uses external LoRa)	SPI, Bluetooth	Integrated Bluetooth; Supports Arduino/Circuit Python	Doesn't have native LoRa; Operates in a different band (2.4 GHz)
STM32WL	Yes	SPI, I2C, UART, ADC, GPIO	MCU + transceiver in same chip; low consumption; LoRaWAN compatible	Cost; Complexity
ESP32	Yes (depends on the version)	Wi-Fi, SPI, I2C, UART, ADC	Wi-Fi + LoRa; cheap; efficient in deep sleep	Initial configuration more complex; sensible to batteries
Arduino UNO	No	UART, I2C, SPI, GPIO, ADC	Simple and cheap; big community; robust	8-bit, small memory; no Wi-Fi; no native support for 3.3 V; no LoRa

2.1.4 Gateway and LoRa Cloud

Gateways are elements responsible for providing communication between the end node (where the sensors are located) and the network server, which communicates with the application server. For such communication to occur, the gateways need to be strategically located in the scenario and insufficient quantity to offer optimal coverage so that all sensors can be connected and transmit information in real time.

Gateways can operate protocol LoRaWAN for LoRa modulation, receiving signals from LoRa devices and transmitting them to a cloud, but they can also work with

Long-Term Evolution (LTE)/4G by connecting to internet via mobile connection and sending data.

There is also Industrial option which are more robust, with larger coverage capacities and frequently projected to operate in severe environmental conditions, these gateways are more expensive than the previous options that were stated.

One of the main obstacles is gateway placement and how it impacts accuracy. To establish greater accuracy of the propagation models, adjustments are made to adapt the average behavior of the propagated signal to the measured data in specific environments [36].

In [37] it is stated that is vastly important to consider possible physical gateway failures, since a LoRaWAN gateway can be compromised, becoming unavailable, or can even be destroyed due to natural or made-man disasters, reducing the resilience and Quality of Service (QoS) of LoRaWAN applications. Damaged antennas and other equipment caused by a disruption of energy supply and others. Therefore, resilient and functional network planning plays an important role in this operation. Hence, it is needed to deploy a gateway to mitigate problems related to dynamic operations.

Values sent through the gateway will then be available on a LoRa cloud where they can be accessed to make further conclusions about quality of air, this will be done via MQTT for connection with the LoRa cloud, MQTT works in a publish/subscribe model that separates both entities using an intermediary, called Broker. The gateway, which receives data and sends it to a LoRa cloud is the publisher while the cloud functions and a subscriber. Most gateways have a packet forwarder which encapsulates LoRaWAN received packages and then forwards them to the server. This communication occurs mostly through User Datagram Protocol (UDP) and uses MQTT for configuration and control.

The network server processes the received packets, validates them and decodifies the payload data. It then communicates with the final application using a specific Application Programming Interface (API), for example Representational State Transfer (REST) or a webhook (via Hyper Text Transfer Protocol Secure (HTTPS)) or even MQTT, communication is done through Transmission Control Protocol/Internet Protocol (TCP/IP), normally with a configured endpoint in the final application.

One example is the The Things Network (TTN) [38] cloud which is one of the most popular platforms for using LoRaWAN which allows users to develop their own IoT solutions with LoRaWAN infrastructure and also uses MQTT for communication and allows for real-time monitoring via its API.

Loriot [39] is also an IoT platform that offers a solution to implement and monitor LoRaWAN networks on a large scale. Some useful resources are the complete management of devices and LoRa networks and supporting different gateway types and LoRaWAN devices.

Kerlink Wanesy [40] is also a different example of a platform that is used in a more business context for operating and managing a LoRaWAN network.

Gateways can be classified as public, private, indoor, outdoor. LoRa gateways can also be multi-channel or single-channel.

Public LoRa gateways are used by providers and can connect thousands of devices scattered across wide areas. Normally installed in public infrastructure of buildings. Private LoRa gateways as provided by enterprises or individual users inside private networks and are controlled directly by the user. Gateways can also differ if installed indoors, which have less coverage and supplied by electricity, or be outdoors, having a higher resistance to hazardous climatic conditions and coverage.

A single-channel LoRa gateway has low cost and is normally ideal for experimental applications and small systems. It is called single-channel for having one only uplink channel at a time and not preferred in complete LoRaWAN networks. On the other hand, multi-channel gateways can support multiple uplink channels at the same time, normally 8 or more, and allow for efficient communication with a higher number of devices. In scalable and complete LoRaWAN networks, this type of gateway is essential and better to use.

2.2 Related Works

This section reviews some relevant and related works done with IoT systems that focus on air quality monitoring. This helps to contextualize the present dissertation within the current state of the art, identifying common approaches, sensor technologies and even communication protocols used in similar systems.

2.2.1 Air quality monitoring in urban environments with a developed system

In [41], air quality monitoring is done by a real time air quality monitoring system, much like what is presented in this dissertation. This project focuses on using multi-parameter air quality monitoring systems that make it able to go to a higher detail level analysis of major pollutants and their sources. Hence, a cost-effective approach for measurement of relevant environmental parameters based on a scalable sensor array with integrated amperometric sensor and infrared gas sensor. This device was then tested in the city and the data it produced was compared with the output data of the local environmental control quality authority stations.

The system, named *OutSense*, includes sensor nodes, wireless routers, server and end devices. Sensor nodes occupy the position of measuring concentration of the main air pollutants and sending acquired data to the *web server*. A mechanism that allowed sleep periods was introduced in the node so as to extend battery life. All sensors went through a warm-up period and then it would start making measurements (operating

mode), this was done in about 30 seconds for amperometric sensors and less to other sensors (humidity, temperature and pressure). The GPS mode is powered on as soon as the measurements start. The system records every value, GPS location and timestamp into a SD Card forming a packet, which then kickstarts communication mode.

If an Internet connection is available, the node contacts the *OutSense* server and immediately sends the pack in JSON format. Otherwise, it waits for a proper internet connection.

The sleep process then starts to the GPS module and all but the amperometric (to prevent reconditioning) where the system enters a low-power state.

Data is then stored in the server, which is composed of 2 main blocks that are responsible for storage and presentation. (i) *Database*, the main block of the server that is responsible for storing the measured data. Every piece of information from a single sensor is stored as a single record. (ii) *Web Server*, which provides the web user interface of the system and receives data from all nodes and stores it in the database. Values obtained for NO sensors had a relative error of ±10%, for CO sensor, these values do not exceed ±15% and for PM_{10} are within an interval of -10% and +15%. This dissertation will however distinguish itself from this project by using LoRa communication and LoraWAN cloud TTN to obtain and make sensor values accessible to the user. This work concluded that low-cost sensors are not as accurate as official data, but they can provide useful indications of air quality in a specific location. Furthermore, the data quality that was provided by the sensor nodes depended significantly on their accuracy of the used low-cost sensors, with the authors solving this limitation by using a scalable sensor array with integrated amperometric and infrared gas sensors. This allowed for higher precision measurements of gas concentrations.

2.2.2 Implementation of LoRa Technology in the Development of Web-Based Air Particulates Monitoring and Advisory System

In [42] a device which transmits data of air particulates to the LoRa gateway, and the network server receives information from the gateway. The web-based system provides information from the gateway. The web-based system application provides the Air Quality Index (AQI) level of health concern, as well as level of $PM_{2.5}$, CO, CO_2. NO_2 and temperature-humidity at a certain location.

This study used four sensors, MICS-4541 for CO and NO_2 detection, CCS811 for CO_2 detection, SHT21 for temperature and humidity detection, and the DSM501A for $PM_{2.5}$ detection and the SX1276 LoRa module. The microcontroller used was an Arduino-Nano.

The developed system is made up by all sensors described above, those detect air particulates and transmit the collected data to LoRa transceivers using LoRa communication technology. The transceivers then send the data to LoRa gateways,

which further forward it to a LoRa network server through TTN. The server stores the collected data and generates advisories. Users can access a web-based application user interface in real time to view the data and the received advisories. This system also allows for monitoring of air particulates quality while providing users with advisories through an accessible web-based application user interface ensuring further collection. Transmission and presentation the air quality information.

A calibration to dust sensors was needed to measure dust, smoke, pollen and pollutants. This was achieved by subjecting the sensor to and either clan air or air with a concentration of a particular matter. Measured values by this sensor reached an all-time high of 255.20 with the equivalent AQI value of 305 (categorized as hazardous), the lower value was 81.09 with an equivalent AQI of 305 (categorized as unhealthy). The air quality in Shaw Boulevard was compared to Addition Hills, which is the first having and AQI value of 142 (categorized as Unhealthy for Sensitive Groups) and the later as having a value of 115 and the same warning.

The highest temperature reached was 35.29 degrees Celsius with a humidity of 49.49 g/m^3. CO reached it biggest value during the morning having 0.05 ppm, CO_2 stayed consistent and NO_2 reached its peak in the morning registering a value of about 0.44 ppm.

This work concluded the monitoring air quality is deeply important to develop better ways to protect human health. It also concluded that LoRa has the capability to have its data range at least 5 Km, but it still depends on the frequency and the location of the sensor device in terms of height and obstructions.

The authors concluded that the research helped safeguard the environment and human health from air pollution by developing and improving tools and methods for measuring and monitoring air particulates, making it able to carry out expose assessments, enhance monitoring capabilities and support public health research. They also concluded that LoRa technology can have its data range reach 5 kilometers, but it was still dependable on the frequency and location of sensor device in terms of height and obstructions. LoRa Technology could be used in transmitting data in real-time. This article also proved LoRa could transmit numeric data from the sensor device in real-time.

2.2.3 Using LoRa Modules to Measure Physical Quantities Describing Air Quality and Their Long-distance Transmission

In [43] authors discuss the possibility of having wireless transfer of physical quantities describing air quality, especially in cities with heavy traffic. Specifically, it acquaints readers with the problematics of carbon dioxide, particulate matter, nitrogen oxides measuring, and data transfer via LoRaWAN network. This system consists on sensors attached to a LoRa module, which sends data to a LoRa gateway every two

minutes. The gateway then resends gathered information to a LoRa server where it is converted to a JSON format and sent to an API client.

The sensors focus on collecting data about particulate matter (PM), CO_2, NO_2, temperature, humidity and concentration. These tests were conducted in a heavy traffic city. And LoRaWAN network was chosen due to its minimal energy for data transmission requirements and because it covers large areas. Individual modules were powered by a battery that depends on the measuring interval, may last several years.

The main objective of this system was to be able to respond to deterioration in air quality, this meaning that hundreds of LoRa modules with sensors would be deployed in the countryside as well as in cities, which would continuously measure the air quality in each place and send the values wirelessly to the gates using LoRa technology. From here, data would be sent to its own server. After evaluation, and in case it exceeded its limit, an instruction would be sent to the reception modules and they would, for example, prohibit the entry of vehicles with internal combustion engines into the given area by changing road markings, or other adjustments in order to prevent further deterioration.

Sensor SPS30 was used to measure particulate matter. Sensirion SCD30 was used to measure concentration of carbon dioxide. MiCS-2714 sensor was used for nitrogen oxide. This system was a raspberry PI 3B+, a LoRa Converter Board and LoRa concentrator board from which the gateway was built. The LoRa server was from Chirpstack. The transmitting elements was an Arduino-UNO.

The authors summarize its work by showing data measured by sensors and from the current traffic situation and it concludes it is possible to control traffic using smart traffic lights and thus also regulate the air quality in cities.

2.2.4 Design of Air Quality Monitoring Using LoRaWAN In Human Settlement

In [44] the authors had, as an objective, to create an IoT monitoring system using LoRaWAN for human settlement to plan and develop better habitation that takes into consideration the quality of air. The proposed system used a WisBlock 4 by RAKWireless and Bosch BME680 based sensor module to measure temperature, humidity and barometric pressure. The design was then evaluated by measuring LoRa network's coverage and response time of monitoring the website to display the air quality data. It was then found that LoRa covered area has a stable Received Signal Strength Indicator (RSSI) of about -100 dBm and a signal to noise ratio (SNR) between 0.8 and 8.2 dB. The average response time for the website to display data was about 580 ms.

One of the most important parameters was the AQI which is based on the concentration of pollutants present in the air in a given location, AQI does not have an assigned unit. This study's proposed system has 3 layers, a physical layer, a network layer and an application layer. In the first, there were sensor-based end devices to capture and collect information from the environment. The network layer was responsible

for connecting all smart objects, network devices and servers. Furthermore, it also served to transmit and process sensor data. Lastly, an application layer is responsible for delivering certain devices of the application to the user. Data from the environment received by the sensor was transmitted to the central server through a gateway.

Used end devices like RAK5005-O (the main board that can support dozens of CPUs, sensors and interface circuit boards), RAK1906, a 4-in1 digital sensor board which consists of temperature, humidity, gas and pressure sensor and RAK4630, which comprised an nRF52480 MCU and an SX1262 LoRa chip.

The network or transmission layer accepted the information transferred by the physical one and sent it to the application using communication technology. The authors chose the WisGate Developer DO (RAK7246) which consisted on a Raspberry Pi Zero W, a RAK2246 Pi HAT LPWAN Concentrator module. The built-in RAK2246 Pi HAT used a SX1308 chip from Semtech, a powerful LoRa digital processing engine.

The Network Server chosen was an open IoT infrastructure supported by the LoRaWAN, the secure messaging protocol and widely used. The LoRa Gateway was connected to TTN via internet. Data capture by sensors and end devices was transmitted to TTN and the payload was needed to be formatted on uplink payload of the TTN so that binary data was converted to human readable fields.

The application layer served as an interface for users to manage and get worthy information from IoT system. In this layer, IoT application and data analytics are deployed. Firebase was used as the database to save the transmitted data from TTN and the web-based dashboard used for displaying the AQI with a color code ranging from Green to Brown representing the classifications Good to Dangerous, respectively.

This study placed the gateway about 140-240 meters away from the end devices. The results showed an average value of 61.65 AQI level which was classified as Yellow (Ranging from 51 to 100) which meant the air quality was moderate and it had no effect on human or animal health but affected sensitive plant and aesthetic value. An average pressure of 1005.3 kPa, temperature of 31.2 °C and humidity of 67.69%.

Table 2-3 helps to summarize all the information taken from all these different studies.

Table 2-3 – Comparisons between all Related Works presented

Reference Work	Objective	Used Sensors	MCU and Hardware	Communication	Results
OutSense [41]	Quality of air monitoring in real-time with detailed analysis of different pollutants	Amperometrics, IR gas sensor, humidity, temperature and pressure sensors.	Sensor node + GPS + SD Card	Internet (when available). No LoRa	Erro ±10% (NO); ±15% (CO); ±10-15% PM10. Low accuracy by utility confirmed
Device to detect air particulates [42]	LoRa system for PM2.5, CO, CO_2, NO_2, T/H detection	MiCS-4541 (CO/NO_2), CCS811 (CO_2); SHT21 (T/H); DSM501A (PM2.5)	Arduino Nano + SX1276 LoRa module	LoRa –> Gateway –> TTN –> Web UI	LoRa with 5 km range (depending on the environment)
Wireless transfer of physical quantities [43]	Urban monitoring with LoRaWAN and automatic response	SPS30 (PM), SCD30 (CO_2), MiCS-2714 (NO_2)	Raspberyy Pi 3B+ (gateway), Arduino UNO (transmissor)	LoRaWAN (Chirpstack)	Quality of air to be used in traffic control. Baterry might last years depending on usage to obtain data
IoT monitoring system [44]	LoRaWAN IoT system to evaluate quality of air in urban environments	BME680 (T/H/Pressão/Gás)	RAK5005-O, RAK1906, RAK4630 (nRF52840 + SX1262)	LoRaWAN –> TTN –> Firebase –> Web Dashboard	RSSI ≈ -100 dBM; SNOR 0.8 – 8.2 dB; average AQI ≈ 61.65 (Moderate). Web response = 580 ms.

3 LoRa Network

This chapter aims to explain the modulation scheme that is LoRa, its structure and the importance of payload. It also refers the differences between the two main ways to activate the devices, OTAA and ABP and the choosing of the Spreading Factor. All these important aspects play a part on how to structure the proposed system.

3.1 CSS Modulation

LoRa is a proprietary spread spectrum modulation scheme that is derivative of Chirp Spread Spectrum (CSS) [45] modulation which trades data rate for sensitivity within a fixed channel bandwidth. LoRa also implements a variable data rate, using orthogonal spreading factors, which allow the system designer to trade data rate for range or power, to optimize network performance in a constant bandwidth. LoRa is a layer 1 implementation (Physical) and is agnostic with higher-layer implementations which allow it to coexist with existing network architectures. LoRa uses the frequency band at 868 MHz (Europe) or 915 MHz (North America).

To better understand spread spectrum, it is first needed to recap the Shannon-Hartley Theorem. In theory, this theorem states the maximum rate at which information can be transmitted over a communications channel of a specified bandwidth in the presence of noise. Therefore, it is concluded that the maximum capacity of said channel results of:

$$\boldsymbol{C = B \times log_2\left(1 + \frac{S}{N}\right)} \tag{3-1}$$

Where:

- *C*, Channel capacity (bit/s)
- *B*, Channel Bandwidth (Hz)
- *S*, Average Received Signal Power (W)
- *N*, Average Noise of Interference Power (W)
- *S/N*, Signal-to-Noise ratio (SNR) expressed as a linear power ratio

In a traditional Direct Sequence Spread Spectrum (DSSS) system, the carrier phase of the transmitter changes in accordance with a code sequence. This process is generally achieved by multiplying the wanted data signal with a chosen spreading code, also known as a chip sequence. The later occurs at a much faster rate than the data signal thus spreading the signal bandwidth beyond the original bandwidth occupied by the original signal.

CSS modulation technique is used due to the fact that it needs relatively low transmission power requirements and inherent robustness from channel degradation mechanisms such as multipath, fading, Doppler and in-band jamming. CSS was,

therefore, adopted by IEEE for Low-Rate Wireless Personal Area Networks (LP-WPANs) for applications that require long range and mobility than what O-QPSK could provide [45].

In LoRa modulation the spreading of the spectrum is achieved by generating a chirp signal that continuously varies in frequency. An advantage of this method is that timing and frequency offsets between transmitter and receiver are equivalent, greatly reducing the complexity of the receiver design. The frequency bandwidth of this chirp is equivalent to the spectral bandwidth of the signal. The wanted data is chipped at a higher data rate and modulated onto the chirp signal.

LoRa Modulation has a scalable bandwidth and frequency. It can be used for both narrowband, frequency hopping and wideband direct sequence applications. Different from existing narrowband or wideband modulation schemes, LoRa can be easily adapted for either mode of operation with only a few simple configuration register changes.

Due to its asynchronous nature, LoRa signal is quite resistant to both in-band and out-of-band interference mechanisms. Since the LoRa symbol period can be longer than the typical short-duration burst it provides excellent immunity to pulsed Amplitude Modulation (AM) interference mechanisms.

The chirp pulse is also relatively broadband and thus LoRa is immune to multipath and fading, making it ideal for use in urban and suburban environments where both mechanisms dominate, such as in the nature of the project of this dissertation where an urban area is to be monitored.

Doppler shift causes a small frequency shift in the LoRa pulse which introduces a relatively negligible shift in the time axis of the baseband signal. This frequency offset tolerance helps mitigate the requirement for tight tolerance reference clock sources. It is ideal for mobile data communication links such as wireless tire-pressure monitoring systems and trackside communications for railroads infrastructure.

For fixed output power and throughput, the link budget of LoRa exceeds that of conventional modulations. When taken into conjunction with the proven robustness to interference and fading mechanisms, this improvement in link budget can easily enhance in range.

LoRa modulation employs orthogonal spreading factors which enable multiple spread signals to be transmitted at the same time and on the same channel without minimal degradation to the RX sensibility. Modulated signals at different spreading factors appear as noise to the target receiver and can be treated as such [45].

3.2 Structure and Payload

LoRa has a maximum payload of 242 bytes, and has fields dedicated to Destiny Address (DA), Source Address (SA) and type of packet it is sending [46]. Packets can be classified as Join Request, sent by end-devices to register in the network, join accept, sent by the server when the end-device is accepted, unconfirmed data up/down which are sent by the end-device to the server and vice-versa, respectively, without any need of having an acknowledge (ACK). If an ACK is needed, packets are classified as confirmed data down/up. Figure 3-1 presents the packet header and how it distributes.

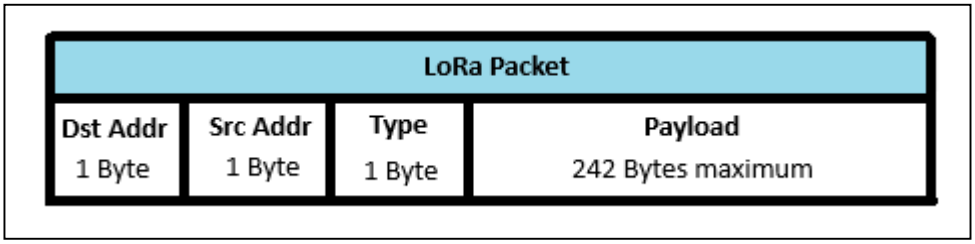

Figure 3-1 - LoRa Packet Header distribution

Knowing how the payload distributes is very important to understand how many bytes it can have for sensors information. Considering values presented at [47], table 3-1 shows the overhead divided by each field, table 3-2 presents the maximum payload per each spreading factor.

Table 3-1 - Overhead LoRaWAN

Field	Size (bytes)
MHDR (MAC Header)	1
DevAddr	4
FCtrl (Frame Control)	1
FCnt (Frame Counter)	2
FPort	1
MIC (Message Integrity Code)	4
TOTAL	**13 bytes**

Table 3-2 - Maximum Payload per Spreading Factor

Spreading Factor	Maximum Payload (bytes)
SF12	51
SF11	51
SF10	51
SF9	115
SF8	222
SF7	222

The six sensors and GPS module presented in the created system for this dissertation are composed the following way. GPS (6 bytes), Dust Sensor (2 bytes), O_2 Sensor (2 bytes), CO_2 Sensor (2 bytes), CO Sensor (2 bytes), NO_2 Sensor (2 bytes) and Temperature and Humidity (4 bytes). Which means having 20 bytes free for sensors and GPS alone. Table 3-3 presents the sensor payload occupation where the 13 bytes overhead also need to be contemplated.

Table 3-3 - Payload Occupation by the sensors

Sensor / Module	Size (bytes)
GPS	6
Dust Sensor (PM2.5; PM10)	2
O_2 Sensor	2
CO_2 Sensor	2
CO Sensor	2
NO_2 Sensor	2
Temperature & Humidity	4
Sensors TOTAL	**20 bytes**
Overhead LoRaWAN	**13 bytes**
TOTAL	**33 bytes**

The sensors were arranged in two different nodes. First to test the gateway's multi-channel capability and next to have even less problems with payload. This resulted in node one having the Dust, CO_2 and Humidity and Temperature sensors resulting in a total payload of 8 bytes. Node 2 therefore had a payload of 12 bytes.

With information taken from [47] table 3-4 present the relationship between SF and data rate with Airtime, Max Duty Cycle and Fair Access policy.

Table 3-4 - Relation between Data Rate and Policies for Both Nodes

Data Rate	Spreading Factor	Bandwidth	Airtime	1% Max Duty Cycle	Fair Access Policy
Node #1	SF7	125 kHz	61.7 ms	6.2 s	486 msg/day
Node #2	SF7	125 kHz	61.7 ms	6.2 s	486 msg/day

Dragino's LG02 gateway only worked with a SF of 7 and was therefore not possible to test all the other spreading factors (8 to 12) [48]. Also taken into consideration was Transmission Power, in Europe this cannot be higher than 14 dBm [49]. Thus, three different powers were tested, 10, 12 and 14 dBm. All these radio tests are present in chapter 5, section 5.3.

Section 5.3 will conclude the best spreading factor and transmitter power chosen to make the tests.

As seen from both figures, in Node#1 the packet would need about 56.6 milliseconds to time on air, with Node #2 this value increases to 61.7 milliseconds. Due to the legal LoRaWAN maximum duty cycle, 1% max duty cycle needs a minimum of 5.7 seconds between any subsequent packet in the same sub band, for Node #2, this value is 6.2 seconds. The last column shows the relationship between data rate and the TTN Fair Access Policy, for Node #1, an airtime of 61.7 ms there is an imposed limit of 530 messages per day. On the other hand, Node #2 with 4 more bytes, has an imposed limit of 486 messages per day. In conclusion, 4 decreases the limit in about 44 messages.

3.3 ABP and OTAA

As evidenced in 2.1.1, there are two ways to activate devices in the network. From one side, OTAA, which is safer and more flexible. The device enters a negotiation and is able to generate a session. The DevEUI, AppEUI and Device Key (DevKey) can all be automatically found by the Microcontroller when it is establishing a connection, in this case, with the LoRa Cloud TTN. OTAA allows for handovers and is more dynamic, on the other hand, introduced overhead and an always mandatory connection with the server.

Then, there is ABP, where devices are pre-configured with an AppSKey, NwkKey, Device Address (DevAddr) with no dynamic activation process. This allows the device to atart transmitting as soon as it is connected. It introduces no initial overhead, and the device becomes server independent. However, there is less security, no key renovation and every parameter change will need to be manually done. Handovers are also not allowed, which means the device needs to stay stationary.

The system built was programmed to work via ABP. This decision was made because the device stayed stationary to measure quality of air in one place only.

However, during the radio tests to understand the strength of the LoRa communication, one thing needed to be done differently, as no handovers were possible, the system needed to be stationary for a couple of seconds in every location for the GPS module to fix the position again.

4 System Architecture and Development

This chapters presents the overall architecture of the system showing not only what composes it, but also how it functions, from the raw data captured by the sensor, all the way to the dashboard. Lastly, it is also shown how a tensor flow could predict values for the sensors based on data collected during the tests.

4.1 System Architecture

In this chapter follows a presentation of the proposed system for real-time monitoring of quality of air. Figure 4-1 presents the system for air monitoring, which is itself composed of six different sensors, a MCU connected through LoRa to a gateway, which is itself connected by LoRaWAN to a cloud that has internet connection and shows, through an API, the dashboard for the final user to get the acquired sensor data.

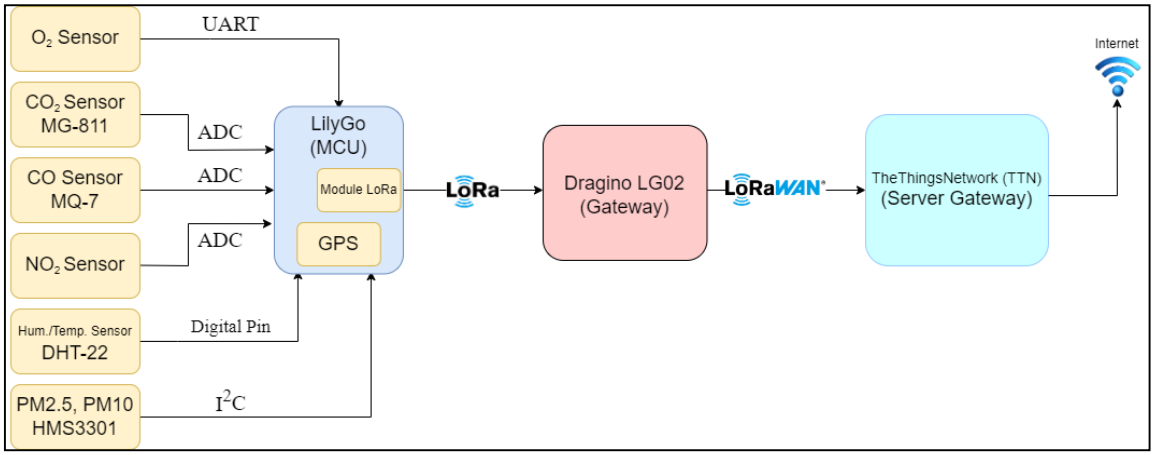

Figure 4-1 - Proposed System

The system was divided into two nodes to study the gateway's capacity to multi-channel and which resulted in also dividing the sensors in two groups of three, each group corresponding to a different node.

Figure 4-2 shows the same created system now divided in two different nodes. Table 4-1 shows the different sensors that were chosen considering their I/O with what was available within the chosen MCU. The node where each sensor is present (#1 for

Node 1 and #2 for Node 2). Other aspects taken into consideration were the economical viewpoint of the sensors, their sensibility, calibration, among others.

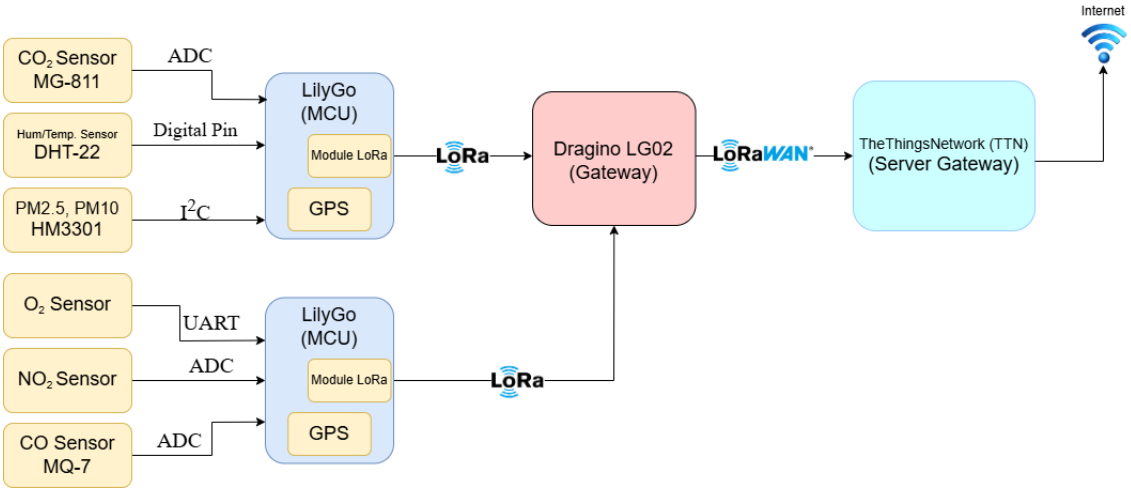

Figure 4-2 - Created System divided into two different Nodes

Table 4-1 - Chosen Quality of air sensors

Sensor	Gateway NODE	Parameter	Values	Interface	Nominal Consumption (mA)
CO_2 MG-811 [50]	#1	CO_2	0 – 50000 ppm	ADC	
MQ-7B [51]	#2	CO	20 – 2000 ppm	ADC	± 70
Compact Analogic Sensor NO_2 [52]	#2	NO_2	0.1 a 10 ppm	ADC	20
O_2 Calibrated [53]	#2	O_2	0 to 25% Volume	UART or I^2C	5
DHT-22 [54]	#1	Humidity and Temperature	**H:** 0.1% RH Accuracy ± 2% RH **T:** 0.1 °C Resolution ± 0.5 °C	Digital Pin	300
HMS3301 [55]	#1	PM2.5; PM10	High Sensibility to µm > 0.3 µm	I^2C	0,15

Node 1 stayed with CO_2 Sensor MG-811, HMS3301 dust sensor and DHT-22 temperature and humidity sensor.

Node 2 stayed with O_2 Sensor, CO Sensor MQ-7B and NO_2 Sensor and the GPS coordinates.

GPS NEO-6M [56] will be already included in the LoRa module, which therefore occupies UART channels. The microcontroller chosen had to include analog inputs for 3 sensors and digital pins for the rest.

Sensors were first tested by connecting them to a breadboard with the Arduino, and tested by using platform Arduino IDE, because the MCU ESP 32 LilyGo® had not been chosen at that moment. Afterwards, the connection was made between sensors and LilyGo® through a breadboard. Sensor calibration was made as it is written in each sensor's datasheet. CO, O_2, NO_2 and HM3301 dust sensors all needed 24 hours working non-stop to calibrate. CO_2 needed 48 hours working non-stop to calibrate.

Figure 4-3 presents the MCU that was used in this project. This MCU, ESP32 Lilygo® [56] was chosen because it already has embedded a GPS receiver and using a battery to better experience load balancing [56]. Table 4-2 shows the MCU main characteristics such as the embedded GPS module and the LoRa modules. The last two rows are saved for the quantity of each different I/O type the ESP 32 Lilygo® has and how many of those are available after all sensors are connected:

Table 4-2 - ESP32 LilyGo® main characteristics

GPS Module	GPS NEO-6M
LoRa Module	SX1276
Quantity and I/O Type	ADCx12; UARTx2; 2xLoRa; I2Cx2; GPSx2
I/O Available after connection:	ADCx8; UARTx0; 1xLoRa; GPSx0
MCU nominal consumption	110 mA
Battery Capacity:	2600 mA

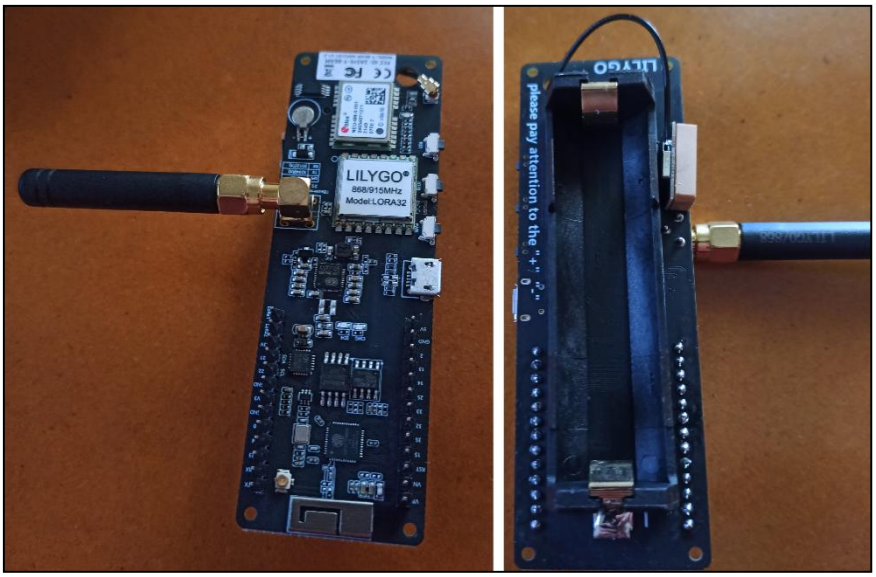

Figure 4-3 – ESP 32 LilyGo®, the MCU that will be used

Figure 4-4 shows the gateway (Dragino LG02) that was used in this project. There was no choice to make in this subject as it was something that the supervisor had

already in his possession and was good enough to use for this dissertation. Table 4-3 shows the main characteristics of this gateway all obtained from [57].

Table 4-3 – Gateway Main characteristics

Number of Channels	2
Interfaces	WiFi, Ethernet port, USB host port
LoRa Modules	SX1276 / SX1278
Sensor Nodes	50 ~ 300

Figure 4-4 - Dragino Gateway LG02

This system works in a specific way presented on figure 4-5. The flowchart shows how each module is accessed when the system is connected.

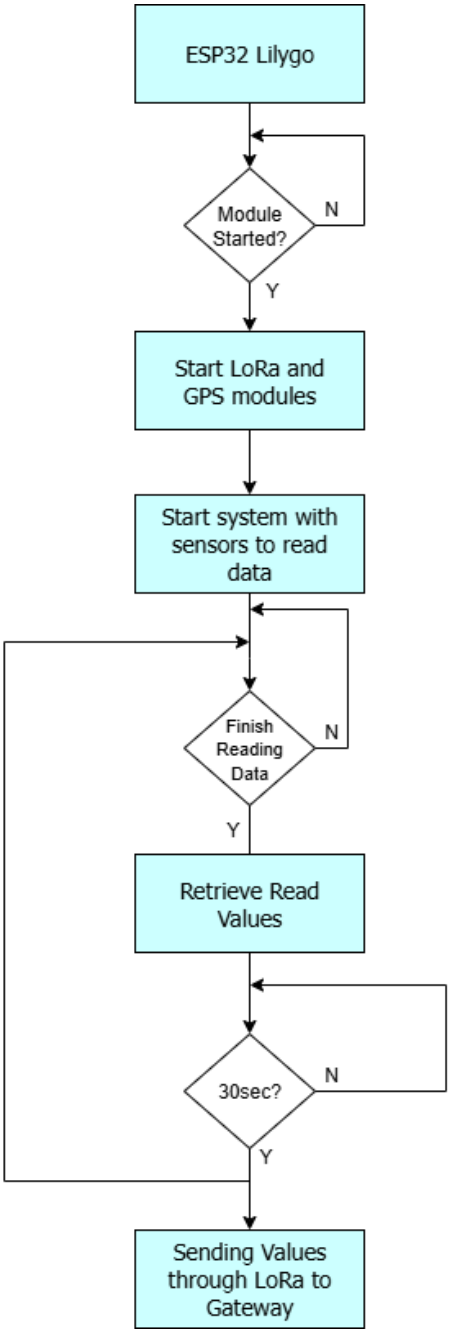

Figure 4-5 - System flowchart after connection

The microcontroller starts with both LoRa and GPS modules when it is started. ESP32 Lilygo® v1.2 comes with the LoRa module disconnected by default, so this step must be done. For this, the code shared in [58] can be of great help, by giving the library already contemplating the start of the Power Management Unit (PMU) module.

After the sensors and the modules are on, the system will start reading quality of air, retrieving values every 30 seconds and, at the same moment, sending them via LoRa to the gateway which serves as a bridge and sends those values in packets directly to the TTN LoRa Cloud.

Figures 4-6, 4-7 and 4-8 show in specific show how each sensor retrieves data and how many bytes are stored in payload for each sensor and GPS modules.

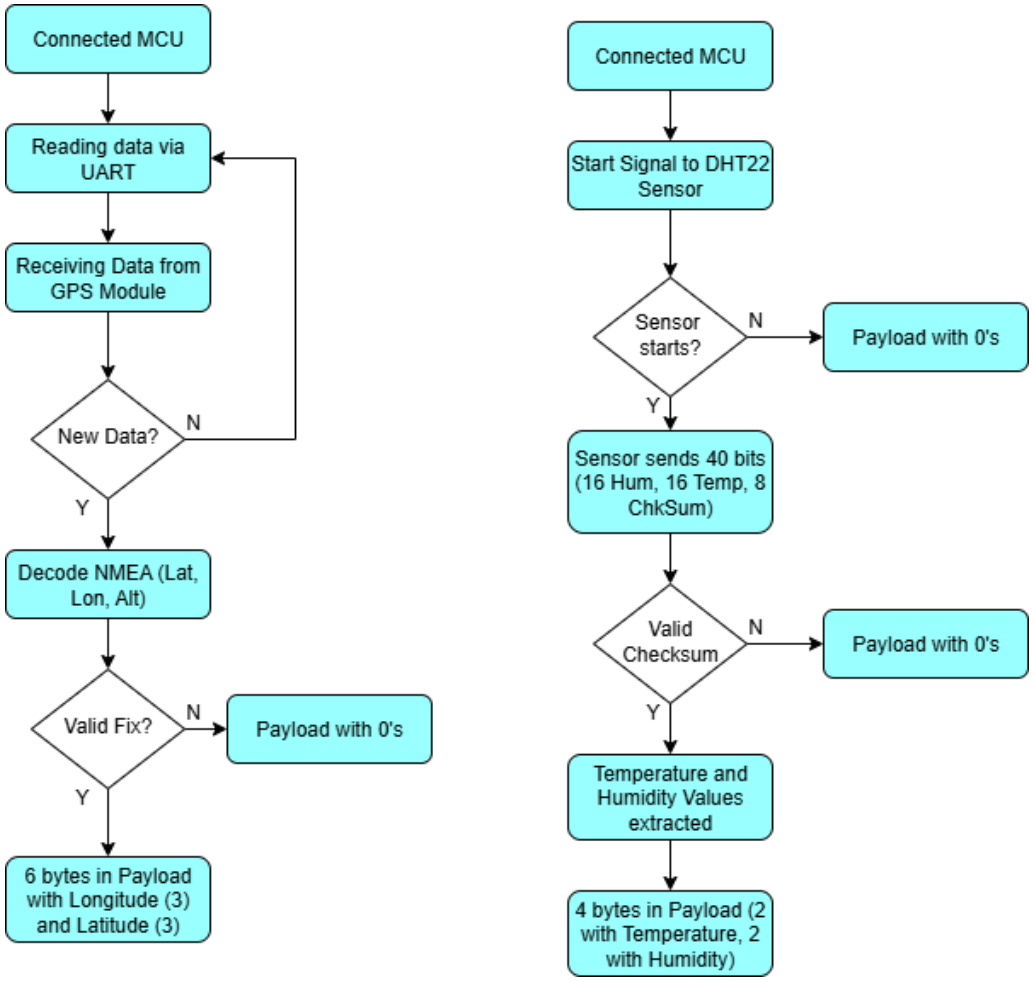

Figure 4-6 - System flowchart on how GPS and DHT22 retrieve data

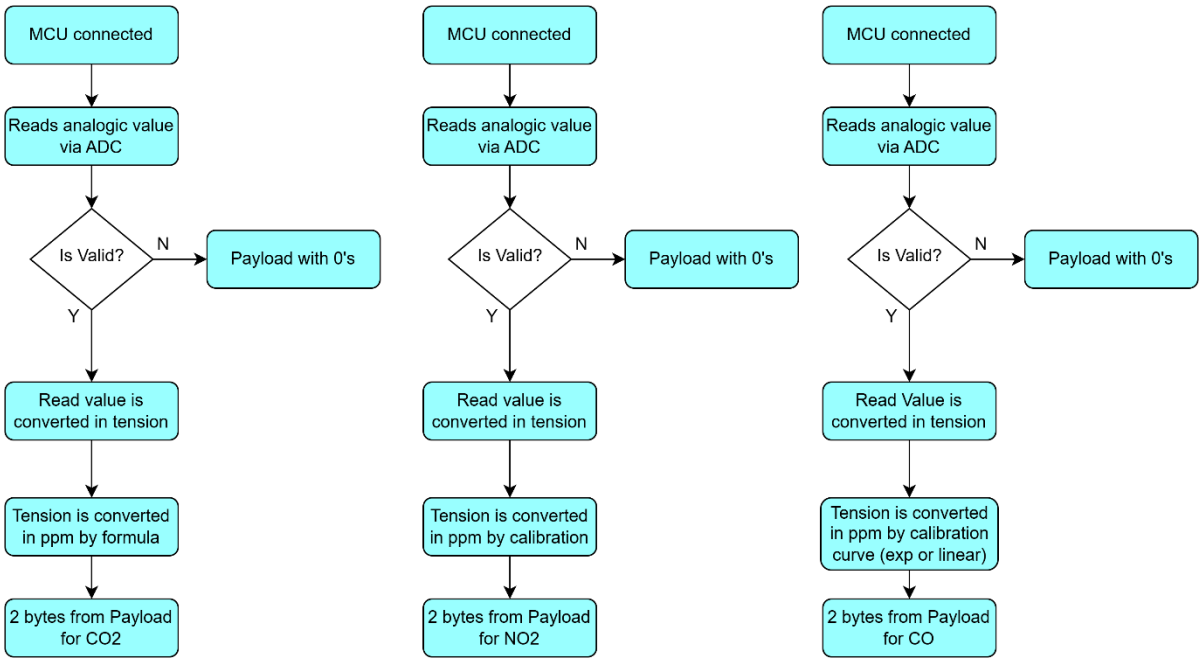

Figure 4-7 - System flowchart on how CO_2, NO2 and CO retrieve data

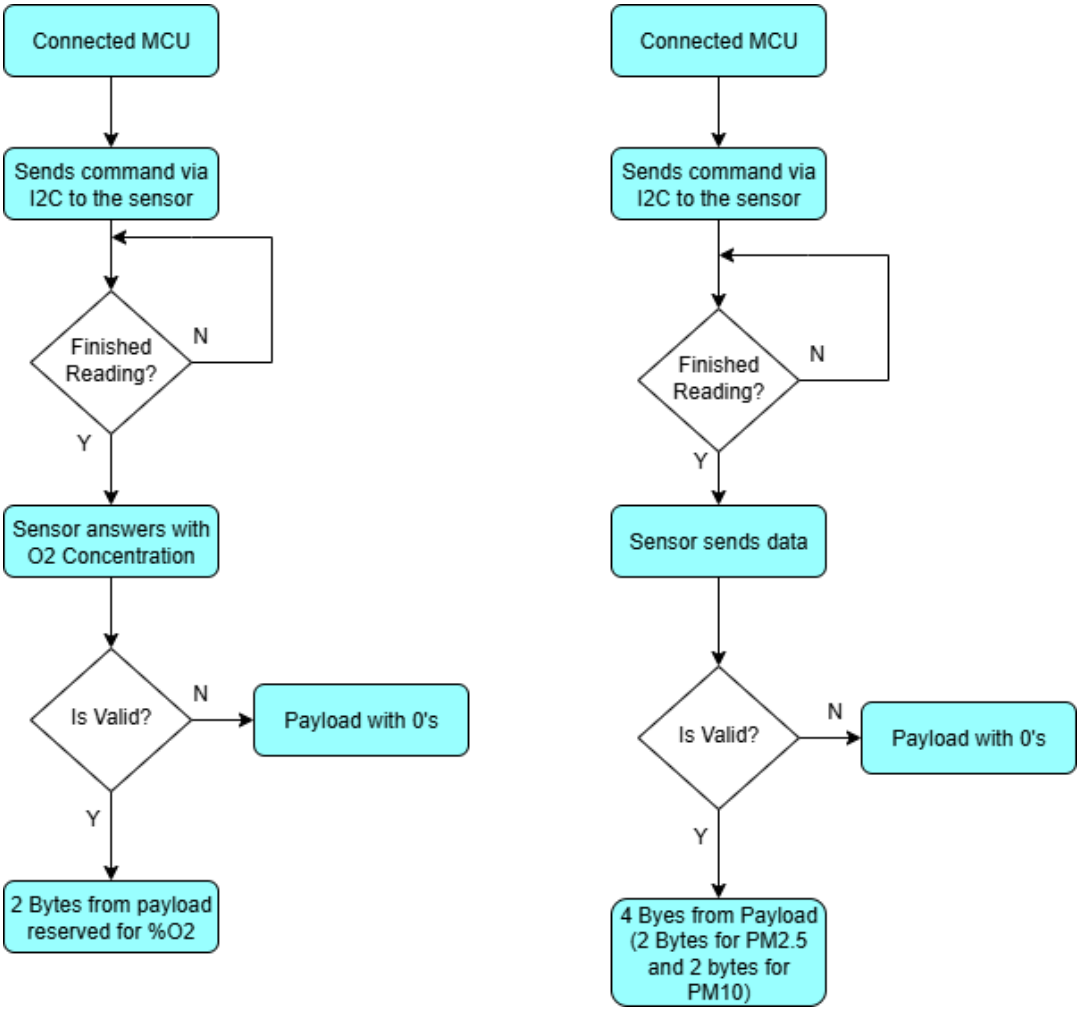

Figure 4-8 - System flowchart on how O_2 and Dust Sensor retrieve data

Following the proposed system, after each MCU collects data, being that node 1's MCU collects data from CO_2, PM2.5, PM10, Humidity and Temperature and node 2's MCU collects data from GPS, O_2, NO_2 and CO, figure 4-9 shows how MCU creates the LoRa packet.

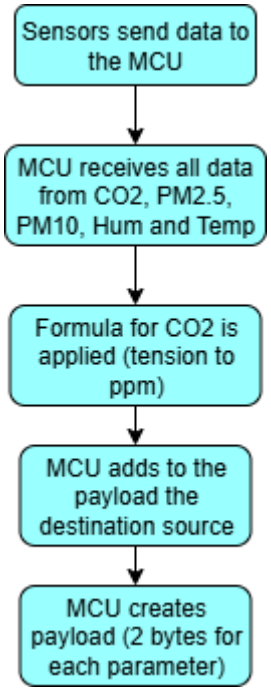

Figure 4-9 – Flowchart on how the MCU create the LoRa payload

After the payload is created and the information ready, the MCU then sends the LoRa packet to the gateway which encapsulates all the packets and sends them via LoRaWAN to the TTN server. Figures 4-10 shows a flowchart with a simplified version on how the gateway works with both nodes, each of them entering a different channel of the gateway. Dragino LG02 allows for multichannel making this possible.

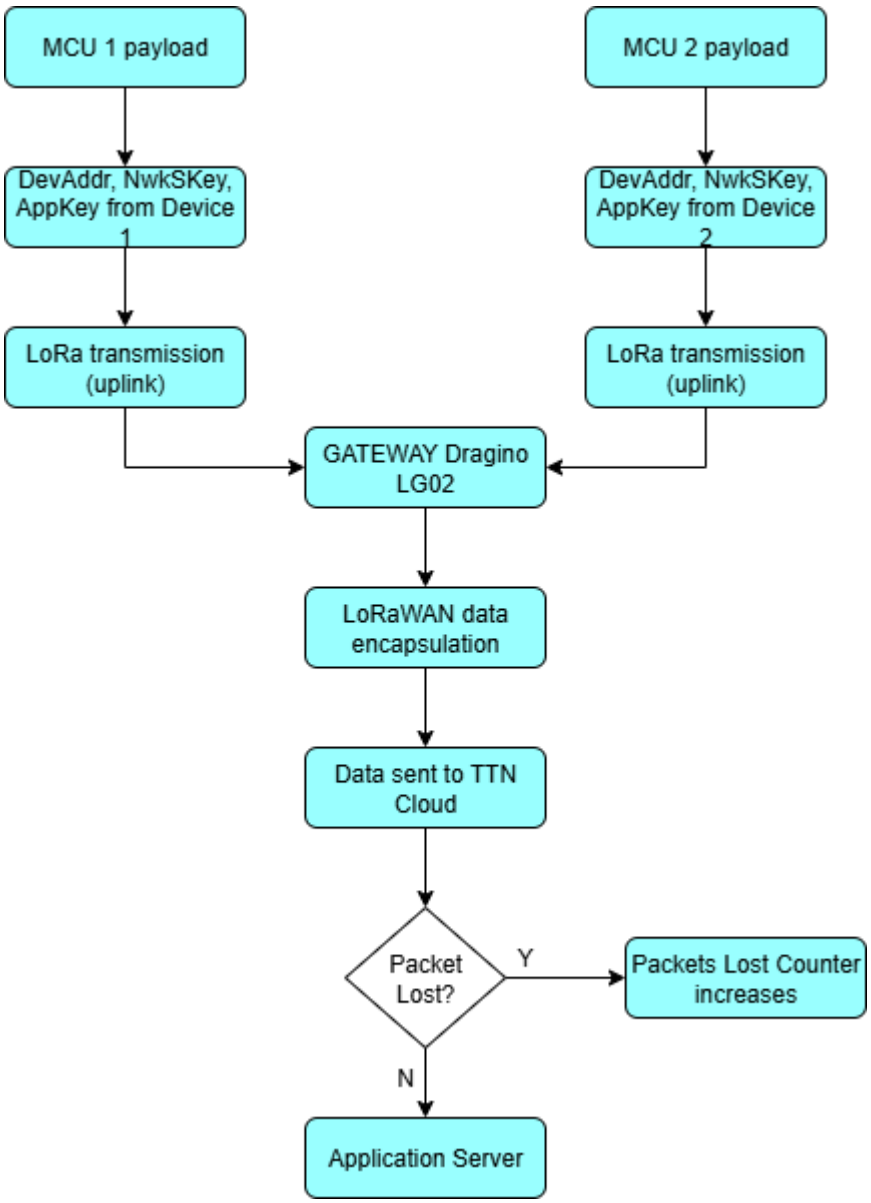

Figure 4-10 – Flowchart shows both nodes sending data to the gateway and then to the TTN server

Following this, the process inside the TTN Cloud Server until it is shown in TTN's own dashboard is shown in Figure 4-11.

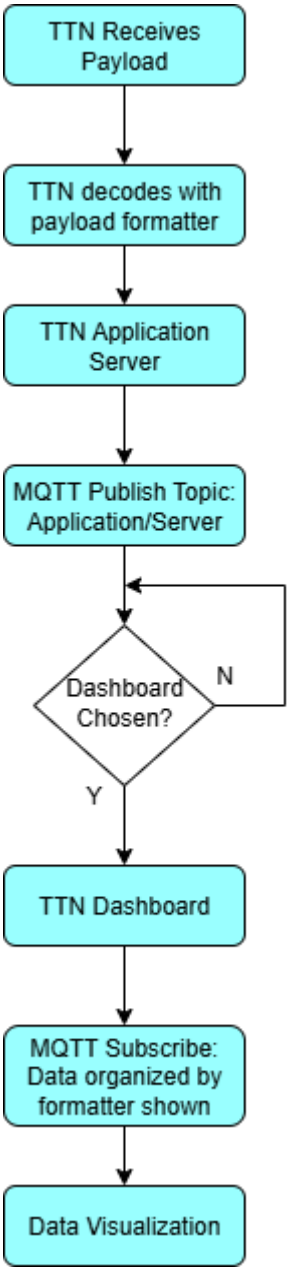

Figure 4-11 – Flowchart which shows the process inside the TTN Cloud until its own Dashboard

4.2 Predictive System

In addition to this study, a predictive system was also created as a way to show how these read values can also be worked into something. Not being at the core of this dissertation, this predictive system is a more simplified version of what can be done with the data acquired. This system evaluates data from six parameters (O_2, CO_2, NO_2, CO, PM2.5, PM10) and works them into a final label with four different classifications (Excellent, Good, Acceptable and Bad). Figure 4-12 shows this system's architecture in all layers, with three inside layers.

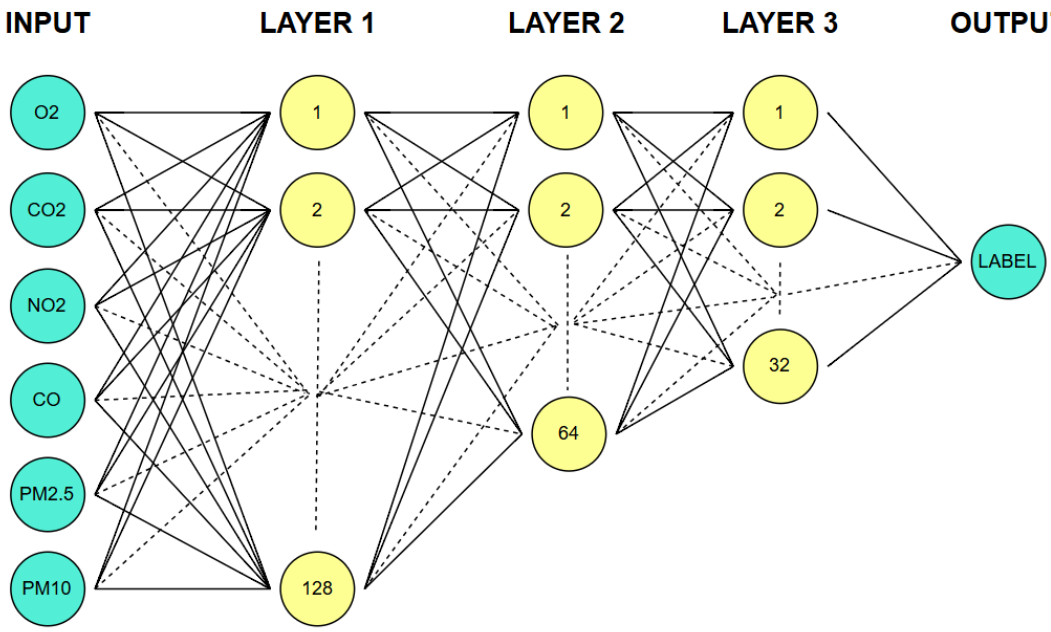

Figure 4-12 – Predictive System Flowchart

5 Results and Analysis

The fifth chapter focuses on presenting all the results the tests provided, for the radio tests (focusing on LoRa only) and the values obtained by the different sensors to fully understand quality of air. It also shows the configuration steps to prepare, not only the gateway, but also the TTN cloud. Lastly, the results of the predictive system created with simplified labels are also shown.

5.1 Gateway Configuration

The gateway's configuration figures inside Annex A. This chapter sums up the entire information.

To configure gateway Dragino, the manual present at [59] worked as the main guide. After booting the Dragino gateway, we can have LAN access to it by searching for the gateway's local IP Address 10.130.1.1 and being connected to gateway's Wi-Fi network, this is shown in figure A-1 present in the Annex A.

Inside the page we connect LoRa and LoRaWAN so that the gateway can connect to the TTN cloud. Figures A-2 and A-3 show the LoRa and LoRaWAN pages displayed with the new parameters. This is where the Spreading Factor and the Transmitting Power can be modified.

There is a need to configure the gateway to work in WLAN, this can be done by modifying the access point, making it work without local Wi-Fi as figure A-4 shows.

Lastly, to configure the gateway to access the TTN LoRa Cloud, figure A-5 shows what to include in the LoRaWAN page, with the other steps inside the TTN cloud being included in the next chapter.

5.2 LoRa Cloud Configuration

This entire configuration is exemplified in figures inside Annex B. This chapter sums up the entire information.

Configuring the TTN cloud starts with creating an account in [60] and choosing a server for the application that needs to be built which can be found in [61]. Figure B-1 shows the choosing server step, as this dissertation was done in Portugal, the closest server to it is located in Dublin, Ireland.

This will then take us to the main console page of the The Things Industries, with all the information as figure B-2 shows. The first thing to be created is the gateway's image inside the TTN Cloud, this is done by configuring the gateway with its Extended Unique Identifier (EUI) and choosing the frequency plan (for this dissertation Europe 868.1 MHz was the chosen plan). Figure B-3 shows the configuration page and figure B-4 the end result with the gateway already created.

Having finished this step, now it is time to add an application to the console, this is connection established in the sensor's direction as the MCU now has a path directed to the TTN Cloud. Figure B-5 shows the configuration page inside the TTN.

With the application configured, now it is time to create an end device, as chapter 3.3 refers, the chosen activation mode was ABP and for that to be implemented, three steps must be programmed into the MCU, such as the NWKSKEY, APPSKEY, DEVADDR. Therefore, the information read by the sensors will be displayed in this page.

As this dissertation also wanted to ascertain the gateway's capacity to multi-channel, a second end device as created with this objective in mind. Figure B-6 shows the list of both end devices; their names refer to the sensors they host.

At the end of this configuration, the source code in Annex C for each of the nodes can be initiated. Figure 5-1 shows a simplified version where the test was only for GPS module.

```
LMIC-Arduino library only support SX1276 Radio ...
LMIC-Arduino library only support SX1276 Radio ...
LMIC-Arduino library only support SX1276 Radio ...
LMIC-Arduino library only support SX1276 Radio ...
Iniciar GPS...
Preparing data...
Enviando payload: 298395: EV_TXCOMPLETE (includes waiting for RX windows)
```

Figure 5-1 – Serial Monitor showing the beginning of the transmission

Figures 5-2 and 5-3 show the transmission on TTN's dashboard after transmission of the packet ends. Figure 5-4 shows both Nodes communicating at the same time showing two different addresses in the same live data.

EVENT DETAILS


```
{
  "name": "as.up.data.forward",
  "time": "2025-09-10T12:59:39.884185430Z",
  "identifiers": [
    {
      "device_ids": {
        "device_id": "eui-70b3d57ed8004314",
        "application_ids": {
          "application_id": "node-1-lilygo"
        },
        "dev_eui": "70B3D57ED8004314",
        "dev_addr": "27FC038A"
      }
```


Figure 5-2 – Node 1 transmission on dashboard's end

Figure 5-3 – Node 2 transmission on dashboard's end

Figure 5-4 – Live data taken from the Application inside TTN Cloud

5.3 Radio Tests

Radio tests are an important part of this dissertation and were done to understand LoRa's response in terms of RSSI and SNR. By changing the transmitter power, it is possible to comprehend the distance the LoRa signal can travel in a city.

The tests were conducted in Barreiro, Portugal, where buildings and hills can prove to be environmental obstacles for the signal to go from the MCU to the gateway.

As explained in section 3.4, only Spreading Factor 7 was tested with three different transmitting powers.

Figures 5-5 and 5-6 show the graphics of both RSSI and SNR relating them to distance with a transmitter power of 10 dBm.

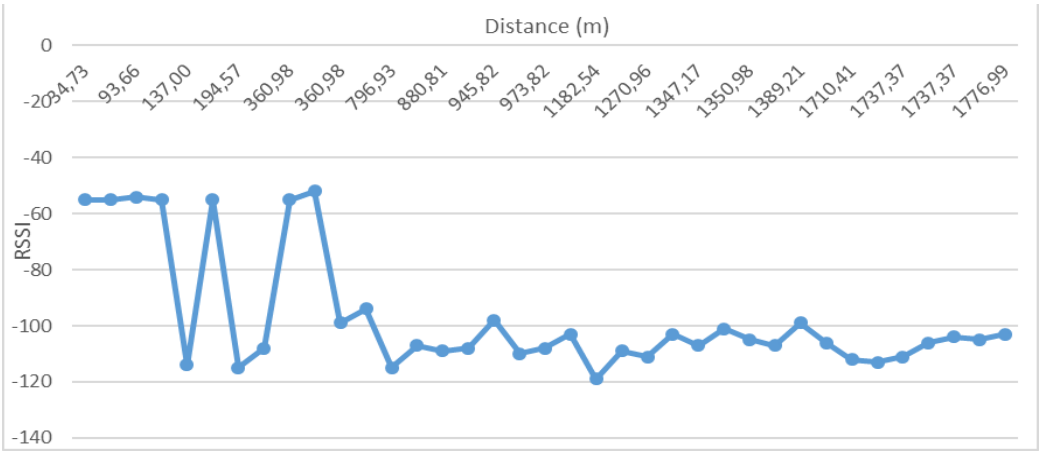

Figure 5-5 - Relation between RSSI and Distance with a transmitter power of 10 dBm

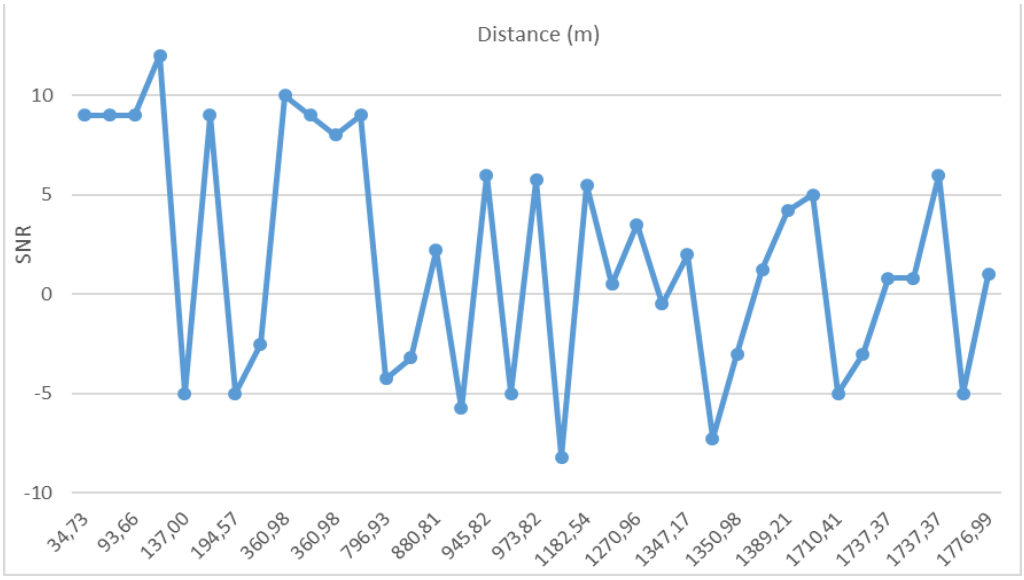

Figure 5-6 - Relation between SNR and Distance with a transmitter power of 10 dBm

The maximum distance with measured values was approximately 1,78 km. For this first radio test, the RSSI had an average value of -98,2 dBm, Figure 5-1 shows the RRSI achieving approximate values of -60 dBm, all while having a positive SNR of 9 dB and even going above 10 dB between 100 and 150 meters. The struggles observed in these first 750 meters in both metrics are derived from environmental obstacles. Around 800 meters the RSSI behavior changes to values considered to be poor (around -100 dBm and -110 dBm), it stays on poor values as the distance increases from there. The SNR however maintains some difference after that threshold but the highest value it achieves is around 6 which is beyond what it had. Table 5-1 shows the average values of both RSSI and SNR in different scopes of distances.

Table 5-1 –RSSI and SNR average values over Distance with a Transmitting Power of around 10 dBm

Metrics	0 – 100 m	100 – 200 m	200 – 500 m	500 – 1 km	1 – 1,78 km
RSSI (dBm)	-54,750	-94,667	-81,600	-107,857	-106,889
SNR (dB)	9,965	4,562	8,164	1,898	1,619

The table shows the best values near the gateway (origin point) and worse results when further (after 500 meters). Between 100 and 200 meters has a slightly worse value than 200 to 500 meters which can be due to the existence of buildings in parts of this area that are more scattered after 200 meters, which will lead to less obstacles and therefore better signal reception. The SNR also follows the same patterns of behavior with having the best results in the same areas as the RSSI does.

In the second radio test the transmitting power was changed from 10 dBm to 12 dBm. The following figures 5-7 and 5-8 show the relation between RSSI and SNR parameters with Distance. Table 5-2 shows the Average values in distance scopes. Before starting this second radio test, the expectation for it was to have the average values better than the first, which meant having a RSSI higher than -98,2 dBm and a SNR higher than 1,58 dB.

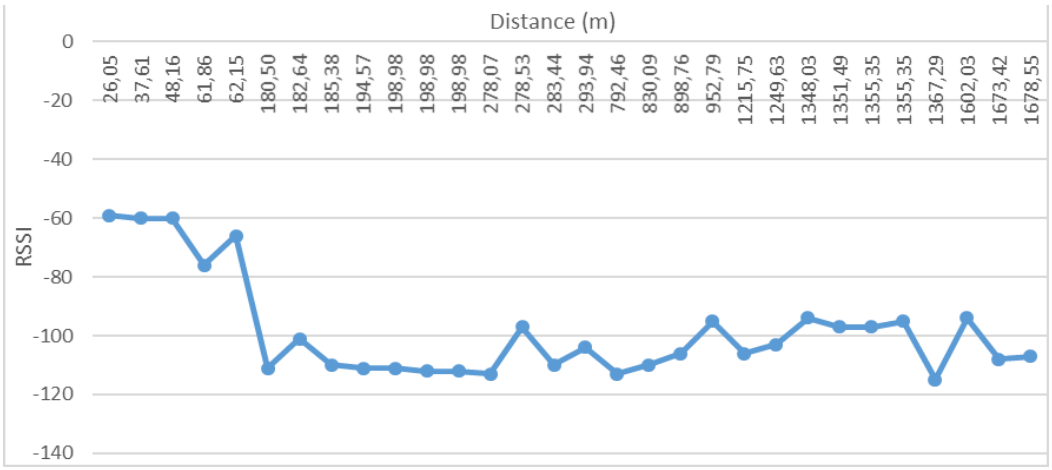

Figure 5-7 - Relation between RSSI and Distance with a transmitting power of 12 dBm

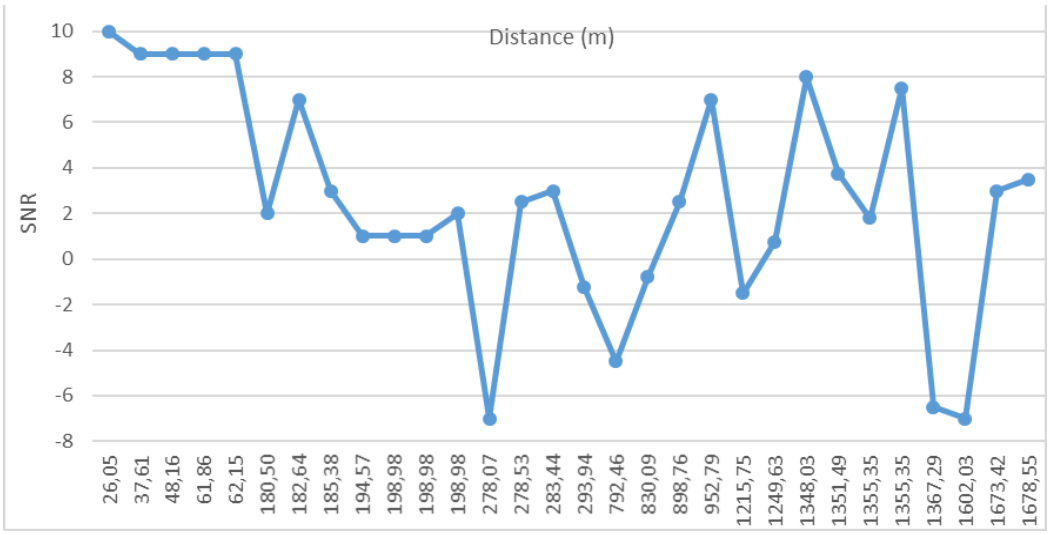

Figure 5-8 - Relation between SNR and Distance with a transmitting power of 12 dBm

Figure 5-7 shows the best efficiency for the LoRa signal is between 0 and 100 meters which is worse when compared to the first test, however, the RSSI maintains its value better when using 12 dBm instead of 10 dBm. SNR graphics show good and positive values in the first 200 meters and then again after 1 Km. SNR also tends to be positive and have a good value if the RSSI is higher or closer than -100. The average RSSI to catch the entire test was -98,4 dBm and the SNR was 2,59 dB, and while the SNR was better than the first test, the RSSI was slightly worse which might be related catching more noise during that test. Table 5-2 shows the different average values of both RSSI and SNR in different distances.

Table 5-2 - RSSI and SNR average values over Distance with a Transmitting Power of 12 dBm

Metrics	0 – 100 m	100 – 200 m	200 – 500 m	500 – 1 km	1 – 1,68 km
RSSI (dBm)	-64,200	-109,714	-106	-106	-101,600
SNR (dB)	9,219	2,996	0,722	2,998	3,497

The second table shows the best values near the gateway and the worst between 100 and 200 meters for the same reason given for the first time. The SNR has the best value between 0 and 100 meters just like the RSSI, but differently from the other metric, the worst results were not on the same scope.

For the third and last radio test, the transmitting power was changed to 14 dBm. Figures 5-9 and 5-10 show the relation between RSSI and SNR with distance. It is expected to get better results for both RSSI and SNR, not only when compared to 12 dBm, but also to 10 dBm.

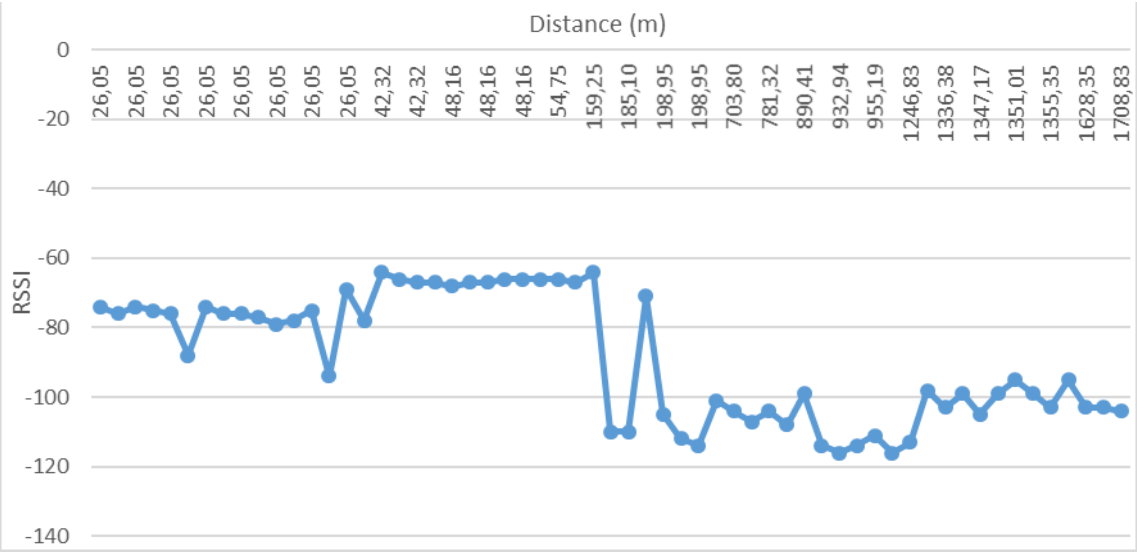

Figure 5-9 - Relation between RSSI and Distance with Transmitting Power of 14 dBm

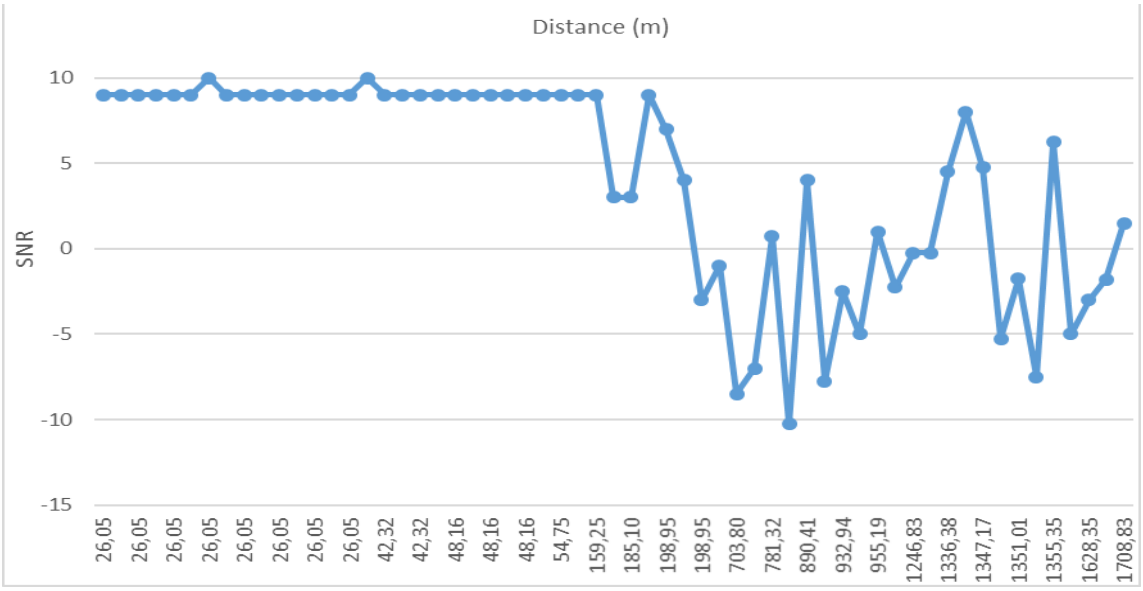

Figure 5-10 - Relation between SNR and Distance with Transmitting Power of 14 dBm

It is possible to observe excellent RSSI values (around -60 dBm) for the first 160 meters, same with SNR (mostly 9 dB), beyond this threshold, the signal loses its strength, maintaining -100 dBm to -120 dBm values in the RSSI metric. The SNR graphics is slightly different but still oscillating between 5 dB and -5 dB. For the entire test, the average RSSI value was -88.73 dBm and the average SNR value was 4.2 dB which are better than both predecessor tests. Table 5-3 shows the average values on both metrics in different distance scopes.

Table 5-3 - RSSI and SNR average values over Distance with a Transmitting Power of 14 dBm

Metrics	0 – 100 m	100 – 200 m	500 – 1 km	1 – 1,7 km
RSSI (dBm)	-72,714	-98	-107,8	-102,5
SNR (dB)	9,08	6,005	-1,405	2,127

The best result was achieved close to the gateway's original location and even though this result is worse in strength that both predecessors, the whole test had between values on average. SNR behaves the same way as RSSI. In both the worse results are registered between 500 meters and 1 kilometer, which also had buildings and hills that can impact the noise and the strength of the signal.

All these radio tests allow the decision to use a transmitting power of 14 dBm and a spreading factor of 7 during the sensor tests that will be referred and studied upon in the next subchapter.

5.4 Sensor Tests

This subchapter offers the visual representation of the values read in all the sensors. Some of the sensors suffered more changes than others, which was also

expected since the study environment was deemed almost perfect. But even though all of that, it still allows for some important conclusions regarding all the results, sensors and even the output interface behavior with the MCU.

The first sensor, the CO_2 sensor, was expected to have around 400 ppm concentration in the air, which is its normal. Figure 5-11 shows the graphic related to it while figure 5-12 shows the theoretical value for CO_2.

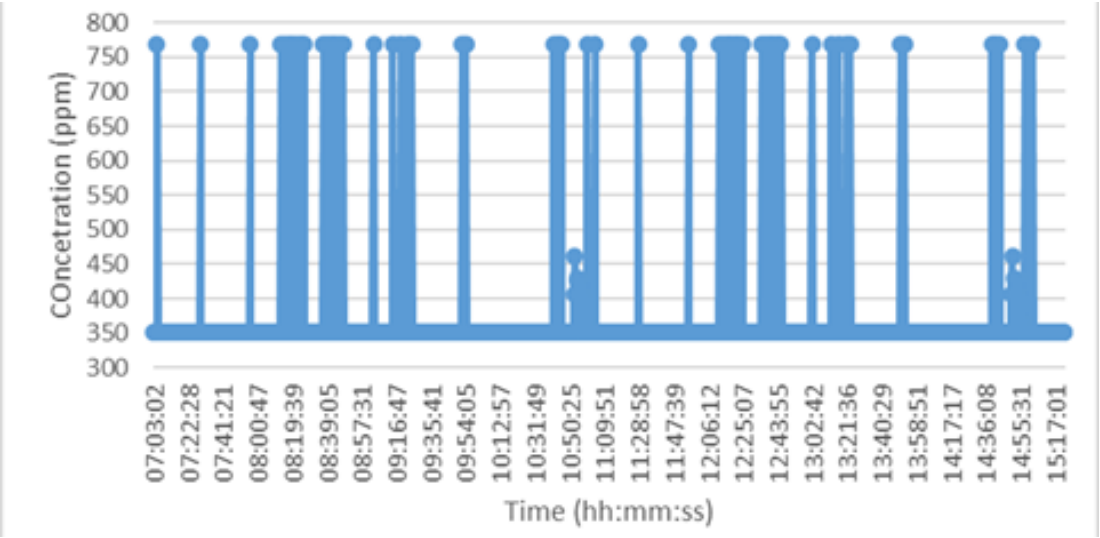

Figure 5-11 - CO2 concentration throughout the test

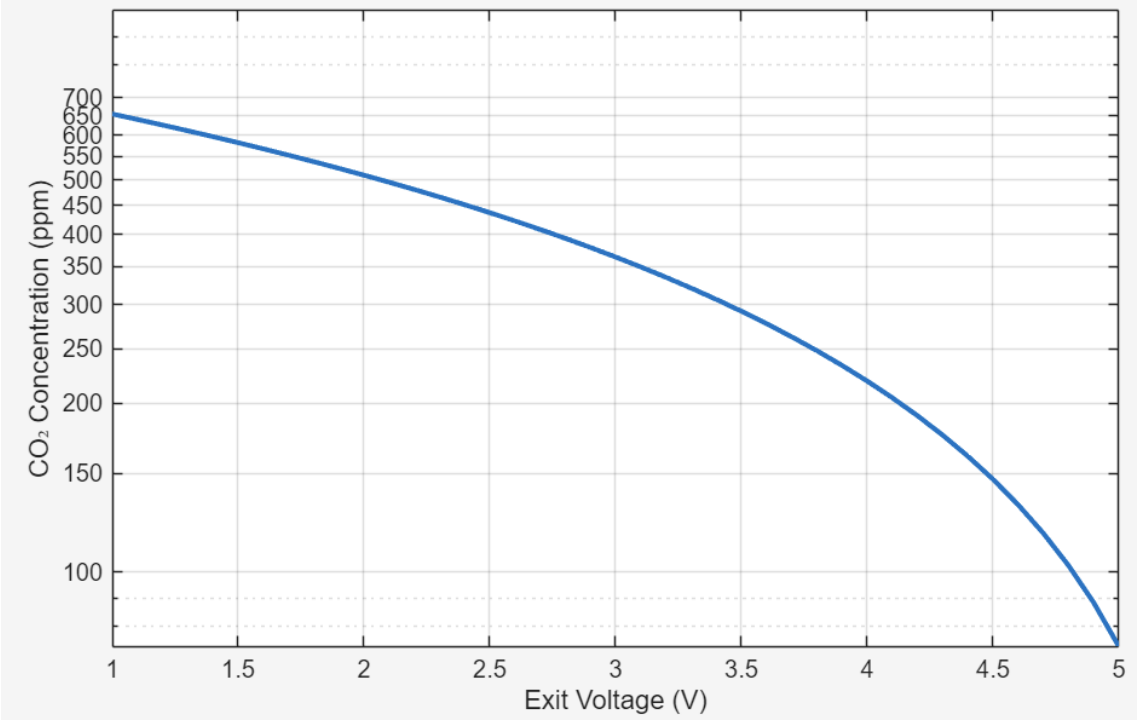

Figure 5-12 - CO2 concentration while varying voltage

Figure 5-11 shows a particular behavior as the CO_2 did not behave in an expected way. This happened due to a combination of hardware and signal-condition limitations. The sensor's output voltage may have exceeded the usable input range of the ADC (0 – 3.3 V) which in turn created signal saturation and errors in conversion. Because the

system also relied on a linear mapping o ADC values any input exceeding the limit where cut which caused reading to converge toward the static value shown. Additionally, the chosen acquisition time ended up introducing delays which prevented the sensor from stabilizing properly, further contributing to unreliable measurements.

Figure 5-12 shows the theoretical value [62], that should be presented after using a voltage divider, which would be approximately 350 ppm (around 3.1 V) which is very closely related to the values the sensor was measuring and also a good concentration of carbon dioxide in the atmosphere.

Secondly, the O_2 sensor also produced values ranging between 20% and 21% which are the normal percentages for this type of gas in the atmosphere. Figure 5-13 shows the graphic of percentage that the sensor read throughout the test. There isn't a theoretical value given this is a UART/I^{2}C sensor, the value the sensor presents is calculated automatically inside it [63].

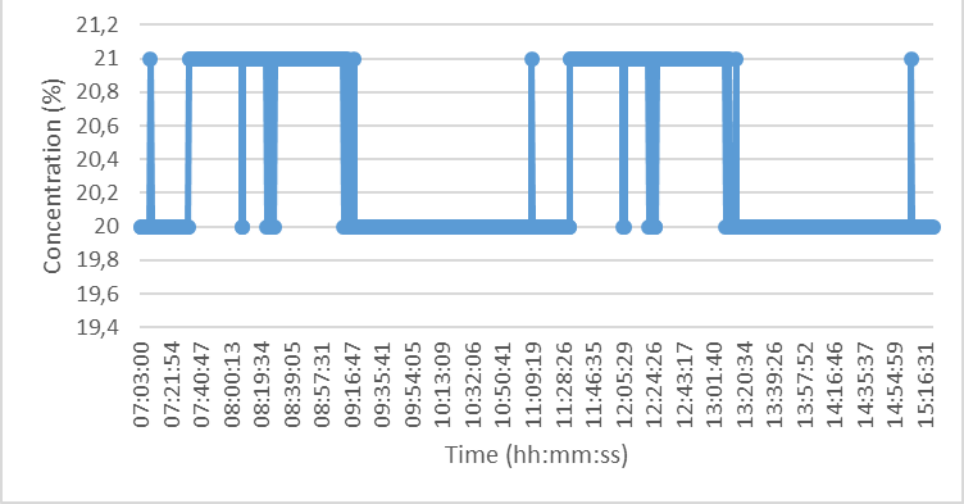

Figure 5-13 - O^2 Concentration throughout the test

From this graphic it is possible to conclude that the O^2 always measured values ranging from 20 to 21 in percentage which correlates to a good concentration of oxygen in the atmosphere, the average Oxygen value was of 20,4% and the standard deviation 0,49%.

The next sensor was divided in three different graphics, in both of those two colors are presented, the blue color represents PM10 and the red one PM2.5. All the conclusions taken from the graphics will then be presented in the form of a table to further summarize the information and lastly conclusions will be drawn from there. Figures 5-14, 5-15 and 5-16 present the amount of particulate matter in the air for PM10 and PM2.5.

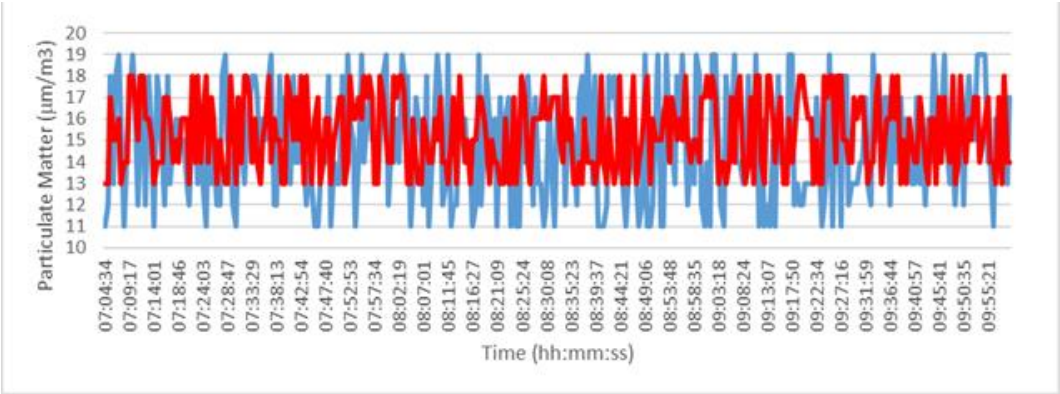

Figure 5-14 - Particulate matter volume in the air from 7am to 10am for PM10 (in blue) and PM2.5 (in red)

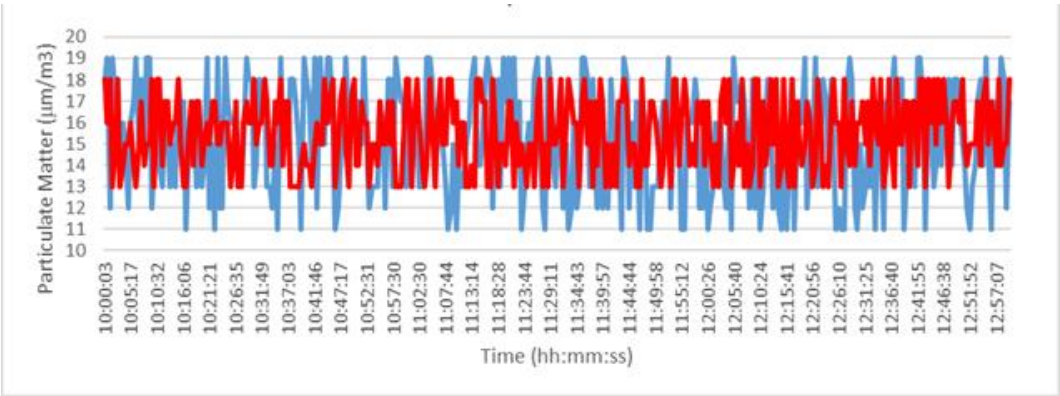

Figure 5-15 - Particulate matter volume in the air from 10am to 1pm for PM10 (in blue) and PM2.5 (in red)

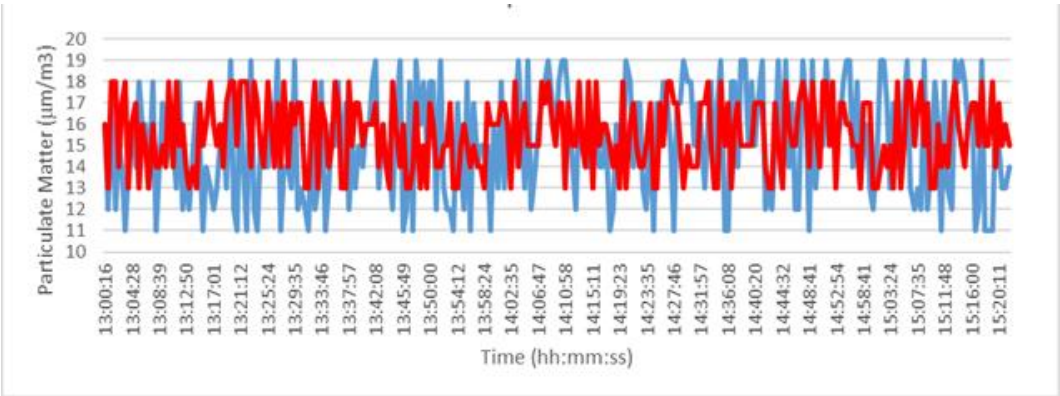

Figure 5-16 - Particulate Matter volume in the air from 1pm to 3pm for PM10 (in blue) and PM2.5 (in red)

All graphics show the changes in both PM10 and PM2.5 throughout the day, however the best way to talk about all these changes is presented in table 5-4 with average values and standard deviation for these periods.

Table 5-4 - Average and standard deviation for both PM10 and PM2.5 throughout the test

Time of Day (Hours)	Particulate Matter Sensor			
	PM10		PM2.5 (µm)	
	Average (µm)	Standard Deviation (µm)	Average (µm)	Standard Deviation (µm)

7am – 10am	14,79	2,544	15,36	1,727
10am – 1pm	15,23	2,553	15,51	1,738
1pm – 3pm	15,09	2,564	15,54	1,672

Taking from the table, it is possible to conclude that PM10 has a higher particulate matter volume between 10am and 1pm, PM2.5 has the higher volume between 1pm and 3pm. The values also differ a bit more in PM10 than PM2.5 due to a higher standard deviation.

According to thresholds studied in [19], PM10 shouldn't be higher than 40 $\mu m/m^3$ and PM2.5 shouldn't pass the 20 $\mu m/m^3$ threshold, which means both metrics are below and therefore with safe values, however if the guidelines followed are not the European Union ones and, instead the WHO organization [64], then PM2.5 is considered high after 15 $\mu m/m^3$ which means in this test, where the average was 15,4, meant the PM2.5 was considered slightly high.

One of the metrics was closer to real-time values than the other, as in the platform AQI [65] the average for that day in PM10 was 18 $\mu m/m^3$ and the sensor average was 15 $\mu m/m^3$. On the other hand, PM2.5 had an average of 6 $\mu m/m^3$ in the platform and the sensor registered an average of 15,4 $\mu m/m^3$. This is a bigger difference than PM10 but can also be connected to the fact that in the platform there is no site close to the place where the tests were made and therefore, these values work more as an approximation.

The following sensors are presented together because both showed some problems during the test, this can be caused to overflow of the ADC converter, connection and feeding problems inside the sensor itself or something that even went wrong during the tension divider when the system was being built. Both sensors, CO and NO_2 show zero altercation during the entire test, with 1,2 and 0,012 measures values, respectively.

In subchapter 5.5, one of these sensors was even put inside the contaminated environment with cigarette smoke to understand if this behavior was normal due to the "perfect" conditions of the test or not, after no changes happened, it was concluded that both sensors must have had a connection problem, a physical problem with the sensor itself or an overflow of the ADC converter, as both sensors had an ADC I/O. Figures 5-17 and 5-18 present the behavior of both sensors throughout the test.

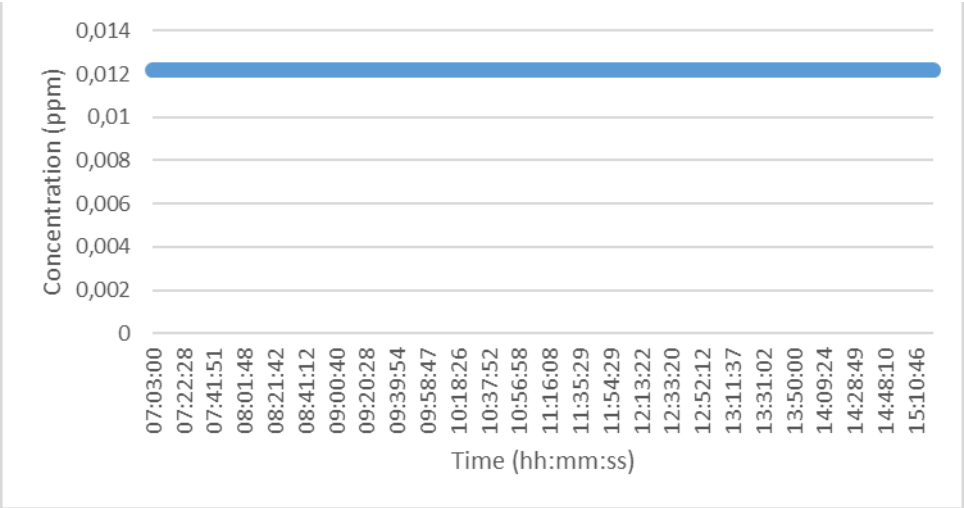

Figure 5-17 - NO2 concentration, in ppm, throughout the test

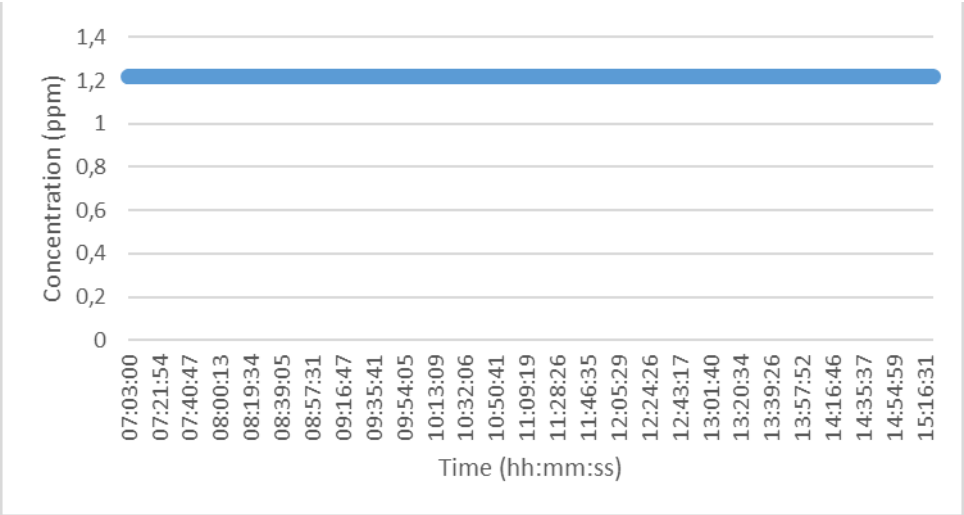

Figure 5-18 - CO concentration, in ppm, throughout the test

From both graphics it is possible to conclude that both sensors have had an overflow in the ADC converter or perhaps the environment was considered too perfect for the sensors to register changes to measured values.

Last two graphics represent the values read regarding temperature and humidity. According to the platform, on the day of the test, the temperature was around 27ºC, this sensor registered a average of 30.6ºC and this was without measuring the night part of day where temperature is normally lower. The relative humidity was around 60%-70% provided by [66], and this is where the sensor underperforms as its average was around 22,4% but the nightly hours are also when humidity is at its highest and therefore, it is expected to have lower results. Figures 5-19 and 5-20 show the relative humidity and temperature respectively.

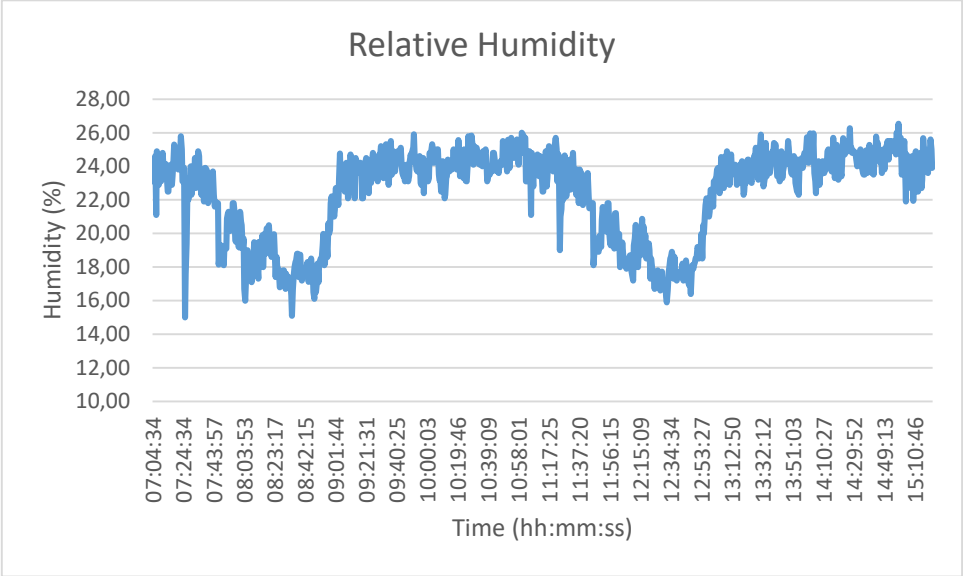

Figure 5-19 - Relative Humidity throughout the test

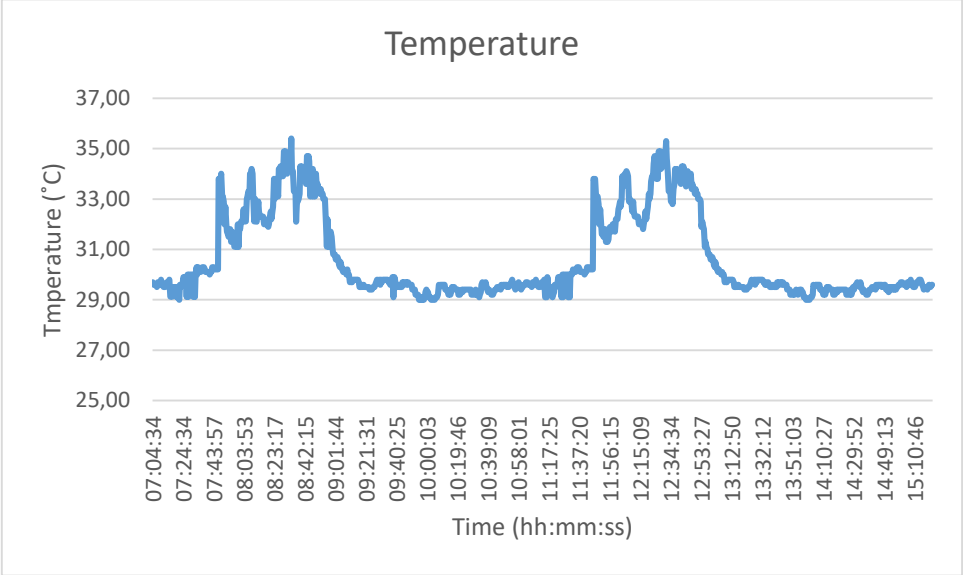

Figure 5-20 - Temperature throughout the test

Both graphics show interesting behavior between 8 am and 9 am with a relatively high temperature and low humidity in a considered relatively cool hour. This happened because the place where the test is set catches its first glimpse of the sun at this hour and it normally accompanied by a big decrease in wind and it windy in this street at this time of the day which therefore consists on having higher temperatures and coincidently lower humidity.

The other period when the temperature is higher again and the humidity is low is during midday which is considered normal due to being the warm period of the day. The average temperature is also higher than normal and all the values almost always close to 30ºC because the test was made close to a house which during this period is still very

warm from the sun of days before and this heat that remains and can also modify some of the temperature values that the sensor reads.

5.5 Cigar Test

This section being called cigar test is purely suggested by the type of test that was made in order to create an environment where the sensors were put on and environment that would definitely change their values, and from this some comparisons could be drawn, namely with the sensors which were considered dubious due to the unchanging nature they had throughout the entire simulation test. For this practical test, the objective was to put the CO, O_2 and CO_2 inside a place that could maintain the cigar smoke inside and therefore make this environment worse than before. Figure 5-21 shows the system with this type of campanula creating the environment. Figure 5-22 focuses on showing the inside of the campanula already blurred with the smoke of the cigar but where the sensors are still seeable.

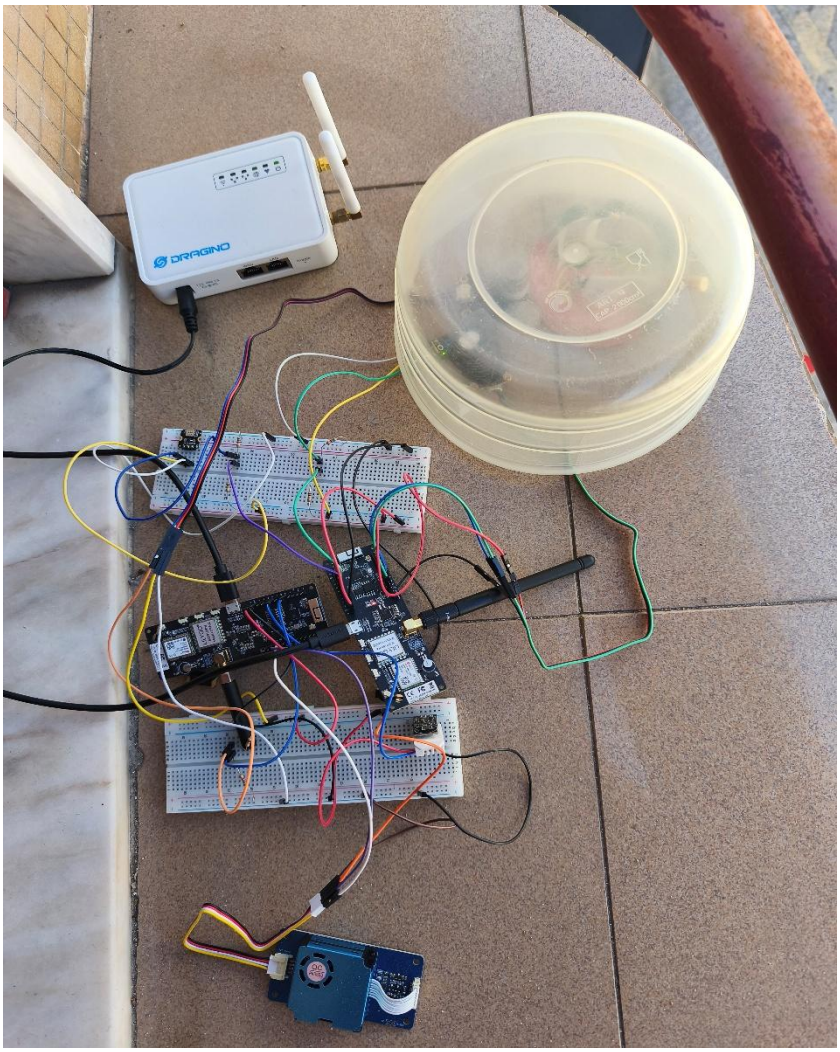

Figure 5-21 - The entire system with the part studied inside the campanula

Figure 5-22 - The sensors inside the campanula with the cigar

The test lasted for about 20 minutes, which was enough to see some slight changes in CO_2 which were expected. It also confirmed the unchanging nature of the CO sensor which could be related to an overflow of the converter or feeding problem of the sensor. The environment created was not perfect in regards of not letting any outside air come through and therefore the changes in oxygen were also not effective. On the other hand, a sensor initially not inside the campanula captured these changes, the dust sensor, with a high increase in both PM10 and PM2.5. Figure 5-23 shows the practical values measured by the CO_2 sensor.

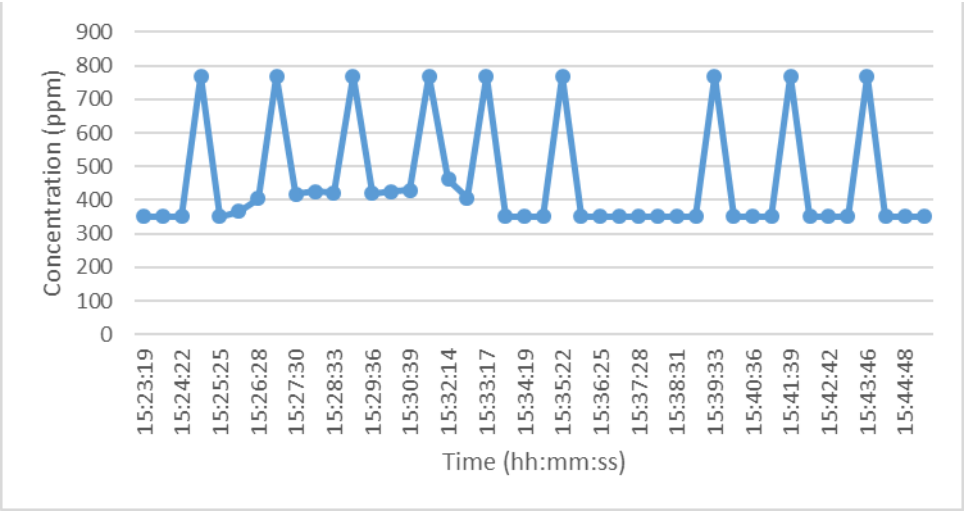

Figure 5-23 - CO2 measured values during practical test

It can be seen from this graphic that the initial differences in the normal 351 ppm exist, with some of the values rising to more than 400 ppm and closer to 500 ppm, however, then sensor becomes unchanged after some time. The average CO_2 value was 456,2 ppm. The value 768 ppm was explained in last subchapter as the moment the sensor needs to clear up to not overflow.

Figure 5-24 shows the O_2 percentage during this practical test.

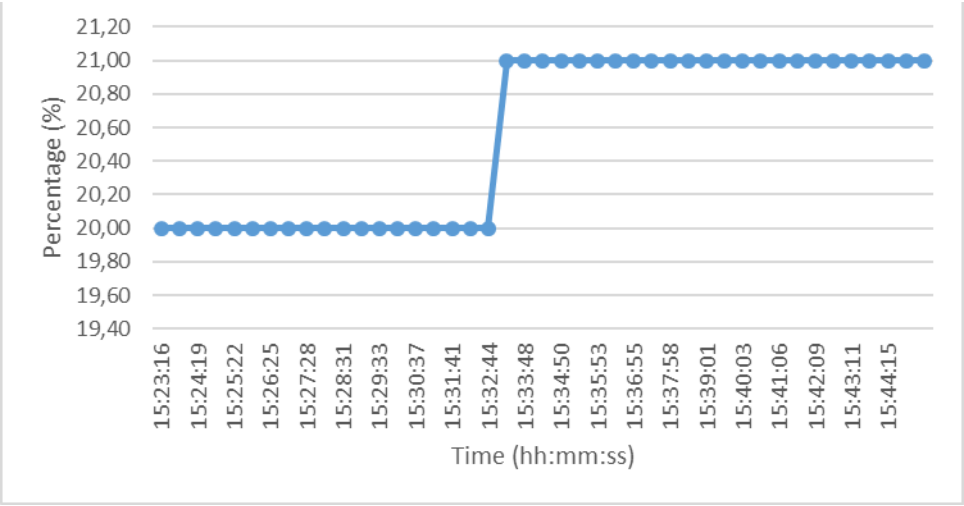

Figure 5-24 - O2 measured values during the practical test

This graphic shows the robustness of the O_2 to an otherwise imperfect campanula, as there is still air flowing, this makes it up for the unchanging nature of the oxygen inside, it is also comprehensible that this sensor would not register any changes due to the fact that the cigar is being consumed at a very low pace, therefore incapable of consuming the oxygen faster and making major changes in the sensor's measurements.

Figure 5-25 presents the PM10 particulate matter volume during this practical test. Without the sensor being inside the campanula, it still registered a higher volume than considered normal and it was deemed important to show.

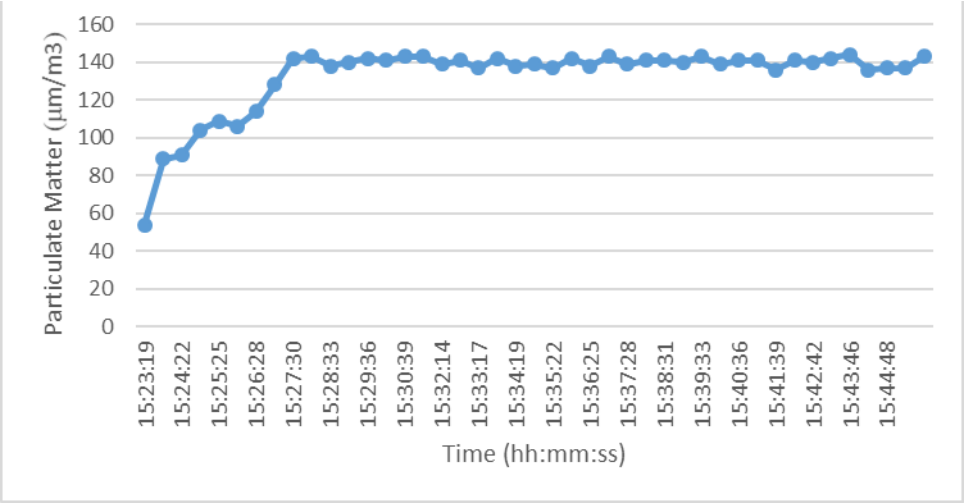

Figure 5-25 - PM10 Particulate Matter Volume during the practical test

PM10 starts the test already observing a higher count of particulate in the air, will values surrounding 50 to 60 µm/m^3, afterwards, these values go up to 140 µm/m^3, this is considered unhealthy for sensitive groups, with an average of 132,7 µm/m^3 proving that people with respiratory problems shouldn't smoke, and although this conclusion has been drawn biologically and chemically for years, it is still interesting and important to see how sensors can allow for this conclusion to also be drawn. PM2.5 also had a big increment of its value, which figure 5-26 presents.

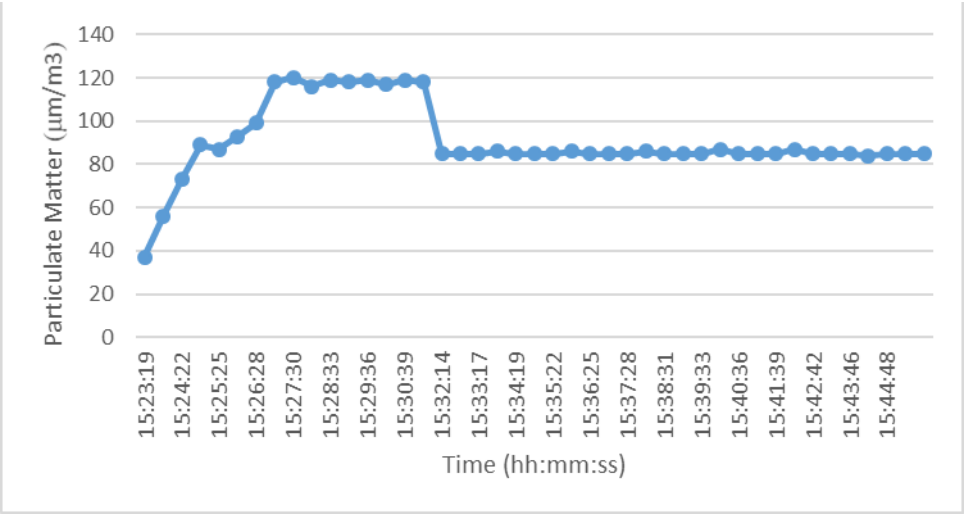

Figure 5-26 - PM2.5 particulate matter volume during the practical test

PM2.5 is far more dangerous than PM10, this has already been proved, as 2.5 µm can move deeper into human lungs and create more health problems. With an average of 90,7 µm/m^3 and a peak value of about 120 µm/m^3, this metric shows a level

considered Moderate but closer to being a risk for sensitive people, however, it is important to note that this moderate level is already considered unsafe for sensitive people as they are weaker and shouldn't be exposed to this quantity of particulate matter in the air.

5.6 Predictive System Results

After applying the architecture, some general guidelines were followed for construction of the following results. According to [67] the guidelines for a 24-hour period for PM2.5 and PM10. For CO, CO_2 are according to [68] and NO_2 according to [69] all shown in Table 5-5.

Table 5-5 – Threshold for some parameters evaluated in the predictive system

LABEL	PM2.5 ($\mu m/m^3$)	PM10 ($\mu m/m^3$)	CO (ppm)	NO_2 (ppm)	CO_2 (ppm)
EXCELLENT	=<15	=<45	=<9	=<0.5	=<400
GOOD	>15 to 25	>45 to 50	>9 to 29	>0.5 to 1	400 to 700
ACCEPTABLE	>25 to 37.5	>50 to 75	>29 to 50	>1 to 5	700 to 1000
BAD	>37.5	>75	>50	>5	>1000

After applying all these rules to a file with all the acquired data from the sensors, the following class distribution was obtained, presented in table 5-6.

Table 5-6 – Distribution of labels along the acquired data

LABEL	%
EXCELLENT	47.29%
GOOD	46.82%
ACCEPTABLE	5.88%
BAD	0.00%

This demonstrates that most of the acquired data values fall into the EXCELLENT and GOOD labels, meaning the conditions of the air in the atmosphere during the test stayed within safe limits. A smaller percentage was deemed ACCEPTABLE and, lastly, no data was labeled as BAD. This labeling can also be used for supervised machine learning training. In conclusion, this is the grounds for the development of predictive models that can classify new air quality measurements automatically.

6 Conclusions and Future Work

6.1 Conclusions

There are lots of conclusions to be drawn from this entire work, regarding the working sensors, LoRa, the gateway and the TTN Cloud. Hence the division into parts.

LoRa/LoRaWAN, was the modulation and communication between MCU and gateway and then between gateway and TTN Cloud. LoRa has a payload limit and therefore there is a need to understand how the sensors work with payload of the measured values before anything else. The communication in spreading factor 7 can reach distances of about 1,8 Km, which is lower what is proposed by the beginning of this study.

The Dragino Gateway, where the main conclusions to be drawn are its impossibility to study beyond the chosen spreading factor 7, as it doesn't work with all others and also the possibility for multi-channel by having at the same time for the main test, two nodes sending values almost at the same time, which made this test battery more efficient and allowed for the separation of sensors between both nodes without having to overload the payload on only one node.

The TTN Cloud was the application chosen to have the values available in real-time, as it was a good cloud with no need for payment, because this is still a very expensive area, this cloud having a base free program was truly unique. However live date lasts for only 24 hours which is something that can damage the test battery if it is more than 1 day.

The sensors have, for the most part, performed well, measuring values close to what is real, considering even the CO and NO_2 sensors which marked their own values but were not outside of the considered range for both metrics even if, throughout the test, they hadn't changed. All ADC sensors are almost always bound to an overflow of the converter, as seen from the unusual behavior they showed, even with CO_2 having its own moments of clearing the converter. All the other sensors worked well and measured normal ranges for all their destined scopes. However, some of these differences could be atoned if the system wasn't as low-cost as it was built, but that would go against the premise of this dissertation which was to build a system from scratch and therefore making it low-cost and available for anyone who would like to create the same system for posterior experiments. Even with this, the system works and proves itself to be good, and, for some sensors, the data is reliable. The predictive system proved to be the ground for what machine learning can do with sensor data acquired even doing something as simple as clarifying some labels for data acquired.

6.2 Future Work

Future work should implement a proper calibration curve based on the sensor's datasheet, using linearization or polynomial regression instead of a single linear mapping. Additionally, using a higher-resolution ADC or an external converter with a wider voltage range would also prevent saturation at 3.3 V.

The CO sensor's performance can be improved by reducing the sampling delay ensuring the analog output is amplified and filtered using an operational amplifier. Calibration should be done with gas values references to correct some misalignments.

The NO_2 sensor requires stabilization and a strict following of its recommended load resistor and amplification circuit. It is also recommended to recalibrate the device using known NO_2 atmosphere concentrations.

Also changing the gateway is the system needs to experiment and possibly deal with different landscapes to have more options for the Spreading Factor. The code itself could also be prepared with the option to automatically choose the best spreading factor.

Using the example of one of the practical work this work relates, this system could be incorporated into semaphores that would work accordingly with it, closing areas where quality of air would become too dangerous in order for the air to clean itself. This would prevent people and cars from being present in those areas, which could lead to an increase in human quality of life.

The predictive system could also be well worked into a dissertation that focuses more on an AI model and machine learning that can create accurate predictions with values for each sensor, as this was not this dissertation's focus, only simplified labelling was presented.

The possibility of following the steps in Annexs A and B to configure gateway and The Things Network cloud can be joined together with the source code present in the github repository at [70] to create a system all while leaving space for anything else to be built on top of this, ensuring new ways to lead this project forward.

Bibliography

[1] World Health Organization, “Ambient (outdoor) Air Quality and Health,” *World Health Organization*, Oct. 24, 2024. https://www.who.int/news-room/fact-sheets/detail/ambient-(outdoor)-air-quality-and-health

[2] S. Sobot *et al*., "Two-Tier UAV-based Low Power Wide Area Networks: A Testbed and Experimentation Study," *2023 6th Conference on Cloud and Internet of Things (CIoT)*, Lisbon, Portugal, 2023, pp. 85-90, doi: 10.1109/CIoT57267.2023.10084912.

[3] Kufakunesu R, Hancke GP, Abu-Mahfouz AM. Collision Avoidance Adaptive Data Rate Algorithm for LoRaWAN. *Future Internet*. 2024; 16(10):380. https://doi.org/10.3390/fi16100380

[4] Fragkopoulos M, Panagiotakis S, Kostakis M, Markakis EK, Astyrakakis N, Malamos A. Experimental Assessment of Common Crucial Factors That Affect LoRaWAN Performance on Suburban and Rural Area Deployments. *Sensors*. 2023; 23(3):1316. https://doi.org/10.3390/s23031316

[5] J. Petäjäjärvi, K. Mikhaylov, M. Hämäläinen and J. Iinatti, "Evaluation of LoRa LPWAN technology for remote health and wellbeing monitoring," *2016 10th International Symposium on Medical Information and Communication Technology (ISMICT)*, Worcester, MA, USA, 2016, pp. 1-5, doi: 10.1109/ISMICT.2016.7498898

[6] Sigfox - Implementing Low-Power Wide-Area Network (LPWAN) Solutions with AWS IoT,” *docs.aws.amazon.com*. https://docs.aws.amazon.com/whitepapers/latest/implementing-lpwan-solutions-with-aws/sigfox.html

[7] Semtech, “Semtech LoRa Technology Overview | Semtech,” *www.semtech.com*. https://www.semtech.com/lora

[8] “3GPP – The Mobile Broadband Standard Partnership Project,” *3GPP*. https://www.3gpp.org

[9] “What Is LTE-M? | IoT Glossary,” *www.emnify.com*. https://www.emnify.com/iot-glossary/lte-m

[10] C. McClelland, “IoT Connectivity - Comparing NB-IoT, LTE-M, LoRa, SigFox, and other LPWAN Technologies,” *IoT For All*, Jun. 30, 2020. https://www.iotforall.com/iot-connectivity-comparison-lora-sigfox-rpma-lpwan-technologies

[11] “LPWAN Connectivity Weightless™ | Ubiik,” *Ubiik*, 2021. https://www.ubiik.com/weightless

[12] L. Li, J. Ren and Q. Zhu, "On the application of LoRa LPWAN technology in Sailing Monitoring System," *2017 13th Annual Conference on Wireless On-demand Network Systems and Services (WONS)*, Jackson, WY, USA, 2017, pp. 77-80, doi: 10.1109/WONS.2017.7888762

[13] B. Hinz, “Beecham Research Report Recognizes LoRaWAN® as the Leading LPWAN Technology for Smart Cities - LoRa Alliance®,” *LoRa Alliance®*, Oct. 30, 2024. https://lora-alliance.org/lora-alliance-press-release/beecham-research-report-recognizes-lorawan-as-the-leading-lpwan-technology-for-smart-cities/

[14] “Zigbee Range: You Must Know The Truth,” *Reolink.com*, 2025. https://reolink.com/blog/zigbee-range/?srsltid=AfmBOooZwlMZ9k375npsc6NhDIauig_7EoyRL4V1e6zcb9PQyiCeTIBh

[15] “Homepage,” *LoRa Alliance®*. https://lora-alliance.org

[16] R. Fujdiak, P. Mlynek, J. Misurec and M. Strajt, "Simulated Coverage Estimation of Single Gateway LoRaWAN Network," *2018 25th International Conference on Systems, Signals and Image Processing (IWSSIP)*, Maribor, Slovenia, 2018, pp. 1-4, doi: 10.1109/IWSSIP.2018.8439232

[17] “Regional Limitations of RF Use in LoRaWAN,” *The Things Network*, 2018. https://www.thethingsnetwork.org/docs/lorawan/regional-limitations-of-rf-use

[18] S. Ali, T. Glass, B. Parr, J. Potgieter and F. Alam, "Low Cost Sensor With IoT LoRaWAN Connectivity and Machine Learning-Based Calibration for Air Pollution Monitoring," in *IEEE Transactions on Instrumentation and Measurement*, vol. 70, pp. 1-11, 2021, Art no. 5500511, doi: 10.1109/TIM.2020.3034109
[19] "Objetivos de qualidade do ar | Agência Portuguesa do Ambiente," *apambiente.pt*. https://apambiente.pt/ar-e-ruido/objetivos-de-qualidade-do-ar
[20] "Os riscos de usar oxigénio sem indicação médica", Oct. 3, 2017. https://www.cpaps.com.br/blog/riscos-oxigenio-sem-indicacao-media
[21] O. US EPA, "What is carbon monoxide?," *www.epa.gov*, Feb. 19, 2019. https://www.epa.gov/indoor-air-quality-iaq/what-carbon-monoxide
[22] EPA, "Basic Information about NO2 ," *US EPA*, Jul. 25, 2023. https://www.epa.gov/no2-pollution/basic-information-about-no2
[23] California Air Resources Board, "Inhalable Particulate Matter and Health (PM2.5 and PM10)," *Ca.gov*, 2015. https://ww2.arb.ca.gov/resources/inhalable-particulate-matter-and-health
[24] "What is an MCU and How do Microcontroller Units Work," *resources.pcb.cadence.com*. https://resources.pcb.cadence.com/blog/2020-what-is-an-mcu-and-how-do-microcontroller-units-work
[25] "STM32 32-Bit Arm® Cortex®-M MCUs," *Mouser.com*, Feb. 21, 2008. https://eu.mouser.com/new/stmicroelectronics/stm32/?srsltid=AfmBOop60WEEQaERGXIupuNHedCqHz-GRCQlFxA8COl3yZCLIz_ZjKH9
[26] "nRF52840 - Nordic Semiconductor," *www.nordicsemi.com*. https://www.nordicsemi.com/Products/nRF52840
[27] "STM32WL Series - STMicroelectronics," *STMicroelectronics*, 2022. https://www.st.com/en/microcontrollers-microprocessors/stm32wl-series.html
[28] Espressif, "ESP32 Overview | Espressif Systems," *www.espressif.com*, 2023. https://www.espressif.com/en/products/socs/esp32
[29] "SX1276," *www.semtech.com*. https://www.semtech.com/products/wireless-rf/lora-connect/sx1276
[30] "Essentials of Microcontroller Use Learning about Peripherals: GPIO," *Renesas*, 2025. https://www.renesas.com/en/support/engineer-school/mcu-programming-peripherals-01-gpio?srsltid=AfmBOorBpJKHhCNIqfPYSx-sdluqiymXGJJhMlyehE6JQ-cJNDDaHius
[31] *Arduino.cc*, 2025. https://docs.arduino.cc/libraries/radiohead/#Compatibility
[32] lady ada, "Introducing Adafruit Feather," *Adafruit Learning System*, May 14, 2017. https://learn.adafruit.com/adafruit-feather/overview
[33] "Introduction to Arduino IDE | Arduino LoRa IoT online tutorial," *cpham.perso.univ-pau.fr*. https://cpham.perso.univ-pau.fr/LORA/HUBIQUITOUS/solution-lab/arduino-lora-tutorial/introduction_arduino_ide/introduction_arduino_ide/
[34] lady ada, "Welcome to CircuitPython!," *Adafruit Learning System*, Dec. 20, 2017. https://learn.adafruit.com/welcome-to-circuitpython
[35] Arduino, "UNO R3 | Arduino Documentation," *docs.arduino.cc*, 2025. https://docs.arduino.cc/hardware/uno-rev3/
[36] Cruz, H.A.O.; Ferreira, S.C.B.; Araújo, J.P.L.; Barros, F.J.B.; Farias, F.S.; Neto, M.C.A.; Tostes, M.E.L.; Nascimento, A.A.; Cavalcante, "G.P.S. Methodology for LoRa Gateway Placement Based on Bio-Inspired Algorithms for a Smart Campus in Wooded Area". *Sensors* 2022, *22*, 6492. https://doi.org/10.3390/s22176492
[37] R. S. Silva, W. Pires, S. L. Correa, A. Oliveira and K. V. Cardoso, "Dynamic resources allocation in non-3GPP IoT networks involving UAVs," *2023 IEEE 97th Vehicular Technology Conference (VTC2023-Spring)*, Florence, Italy, 2023, pp. 1-5, doi: 10.1109/VTC2023-Spring57618.2023.10199941
[38] T. T. Network, "The Things Network," *The Things Network*. https://www.thethingsnetwork.org
[39] "LORIOT - Hybrid Network Management System for Massive IoT," *LORIOT*, 2015. https://loriot.io
[40] "Kerlink, IoT connectivity networks, software and services - Kerlink," *Kerlink*, Oct. 31, 2025. https://www.kerlink.com
[41] M. B. Marinov, I. Topalov, E. Gieva and G. Nikolov, "Air quality monitoring in urban environments," *2016 39th International Spring Seminar on Electronics Technology (ISSE)*, Pilsen, Czech Republic, 2016, pp. 443-448, doi: 10.1109/ISSE.2016.7563237
[42] A. S. Ellares and N. B. Linsangan, "Implementation of LoRa Technology in the Development of Web-Based Air Particulates Monitoring and Advisory System," *2023 IEEE 15th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment, and Management (HNICEM)*, Coron, Palawan, Philippines, 2023, pp. 1-5, doi: 10.1109/HNICEM60674.2023.10589035

[43] P. Marcon *et al*., "Using LoRa Modules to Measure Physical Quantities Describing Air Quality and Their Long-distance Transmission," *2021 Photonics & Electromagnetics Research Symposium (PIERS)*, Hangzhou, China, 2021, pp. 1608-1612, doi: 10.1109/PIERS53385.2021.9695062
[44] T. H. Rochadiani *et al*., "Design of Air Quality Monitoring Using LoRaWAN In Human Settlement," *2022 IEEE 14th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment, and Management (HNICEM)*, Boracay Island, Philippines, 2022, pp. 1-6, doi: 10.1109/HNICEM57413.2022.10109605.
[45] "Application Note 1200.22, LoRa Modulation Basics", Semtech, 2015, pp. 6 - 14.
[46] D. Sousa, D. Hernandez, F. Oliveira, M. Lui, S. Sargento, "A Platform of Unmanned Surface Vehicle Swarms for Real Time Monitoring in Aquaculture Environments", 2019, Sensors. 19. 4695. 10.3390/s19214695
[47] "Airtime calculator for LoRaWAN", *avbentem.github.io*. https://avbentem.github.io/airtime-calculator/ttn/eu868/51
[48] "LG02 LoRa Gateway User Manual LG02/OLG02 LoRa Gateway User Manual Document Version: 1.6.1." Available: https://www.dragino.com/downloads/downloads/LoRa_Gateway/LG02-OLG02/LG02_LoRa_Gateway_User_Manual_v1.6.1.pdf
[49] LoRa — LoRa documentation," *lora.readthedocs.io*. https://lora.readthedocs.io/en/latest
[50] "CO2_Sensor_SKU_SEN0159-DFRobot," *wiki.dfrobot.com*. https://wiki.dfrobot.com/CO2_Sensor_SKU_SEN0159
[51] "MQ 7." Available: https://www.sparkfun.com/datasheets/Sensors/Biometric/MQ-7.pdf
[52] botnroll.com, "Sensor Analógico compacto MEMS de Dióxido de Nitrogénio NO2 (Breakout, 0.1-10ppm) - DFRobot SEN0574," *botnroll.com*, 2025. https://www.botnroll.com/pt/biometricos/5199-sensor-anal-gico-compacto-mems-de-di-xido-de-nitrog-nio-no2-breakout-0-1-10ppm.html
[53] "CO, O2, NH3, H2S, NO2, HCL, H2, PH3, SO2, O3, CL2, HF Gas Sensor Wiki - DFRobot," *wiki.dfrobot.com*. https://wiki.dfrobot.com/SKU_SEN0465toSEN0476_Gravity_Gas_Sensor_Calibrated_I2C_UART
[54] botnroll.com, "DHT 22 – Módulo Sensor de Temperatura e Humidade", *botnroll.com*, 2025. https://www.botnroll.com/pt/temperatura/1585-dht22-modulo-sensor-de-temperatura-e-humidade.html
[55] Shenzhen, "Laser Dust Sensor Data Sheet HM-3300/3600 Dust Sensor Data Sheet Version Number V2.1," 2018. Accessed: Nov. 25, 2025. [Online]. Available: https://files.seeedstudio.com/wiki/Grove-Laser_PM2.5_Sensor-HM3301/res/HM-3300%263600_V2.1.pdf
[56] botnroll.com, "T-Beam Helium: ESP32 LoRa SX1276 868mhz, GPS NEO-6M, Suporte de Bateria 18650 - LILYGO® Q412," *botnroll.com*, 2025. https://www.botnroll.com/pt/esp32/5352-t-beam-helium-esp32-lora-sx1276-868mhz-gps-neo-6m-suporte-de-bateria-18650-lilygo-q412.html
[57] "LG02 / OLG02 OVERVIEW." [Online]. Available: https://www.dragino.com/downloads/downloads/datasheet/EN/Datasheet_LG02_OLG02.pdf
[58] Xinyuan-LilyGO, "GitHub - Xinyuan-LilyGO/LilyGo-LoRa-Series: LILYGO LoRa Series examples," *GitHub*, 2019. https://github.com/Xinyuan-LilyGO/LilyGo-LoRa-Series
[59] "LG02 LoRa Gateway User Manual LG02/OLG02 LoRa Gateway User Manual Document Version: 1.7.0." [Online]. Available: https://www.dragino.com/downloads/downloads/LoRa_Gateway/LG02-OLG02/LG02_LoRa_Gateway_User_Manual_v1.7.0.pdf
[60] The Things Industries, "The LoRaWAN Network server for scale," *Thethingsindustries.com*, 2015. https://www.thethingsindustries.com
[61] *Thethings.network*, 2025. https://console.cloud.thethings.network
[62] "CO2 Sensor SKU:SEN0159" Available: https://mm.digikey.com/Volume0/opasdata/d220001/medias/docus/2552/SEN0159_Web.pdf
[63] "Electrochemical Gas Module." Available: https://dfimg.dfrobot.com/nobody/wiki/5953b463b8712f03d0791e98dd592e78.pdf
[64] World Health Organization, "What are the WHO Air quality guidelines?," *World Health Organization*, Sep. 22, 2021. https://www.who.int/news-room/feature-stories/detail/what-are-the-who-air-quality-guidelines
[65] "Almada, Setubal, Portugal Historical Air Quality Analysis" *aqi.in*, 2025. https://www.aqi.in/dashboard/portugal/setubal/almada/historical-analysis

[66] https://www.ipma.pt, “IPMA - Máxima e mínima,” *Ipma.pt*, 2025. https://www.ipma.pt/pt/agrometeorologia/humidade
[67] “Air Quality Standards database: interactive tool,” *Shinyapps.io*, 2024. https://whoairquality.shinyapps.io/Air_Quality_Standards_V2_2
[68] M. Lemon, “Carbon Dioxide (CO2) vs Carbon Monoxide (CO) – What’s the difference?,” *CO2 Meter*, Jun. 30, 2025. https://www.co2meter.com/blogs/news/1209952-co-and-co2-what-s-the-difference?srsltid=AfmBOoqVsAJfkUo_vb7xZRckP8JyMOBxAbmtayX8AifXKIyWoBuCCjhY#co2-limits
[69] “Nitrogen oxide emissions (NO and NO_2): lowering of the thresholds as of 1st July 2020!,” *Be-atex.com*, Feb. 25, 2020. https://www.be-atex.com/en/news/focus/nitrogen-oxide-emissions-no-and-no2-lowering-thresholds-1st-july-2020
[70] Bernardo0603, “GitHub – Bernardo0603/TFM_Quality-of-Air_System: A repository created for the purpose of sustaining the source code of my master’s degree thesis.,” GitHub, 2025. https://github.com/Bernardo0603/TFM-quality-of-Air_System

ANNEXS

ANNEX A – Gateway Configuration

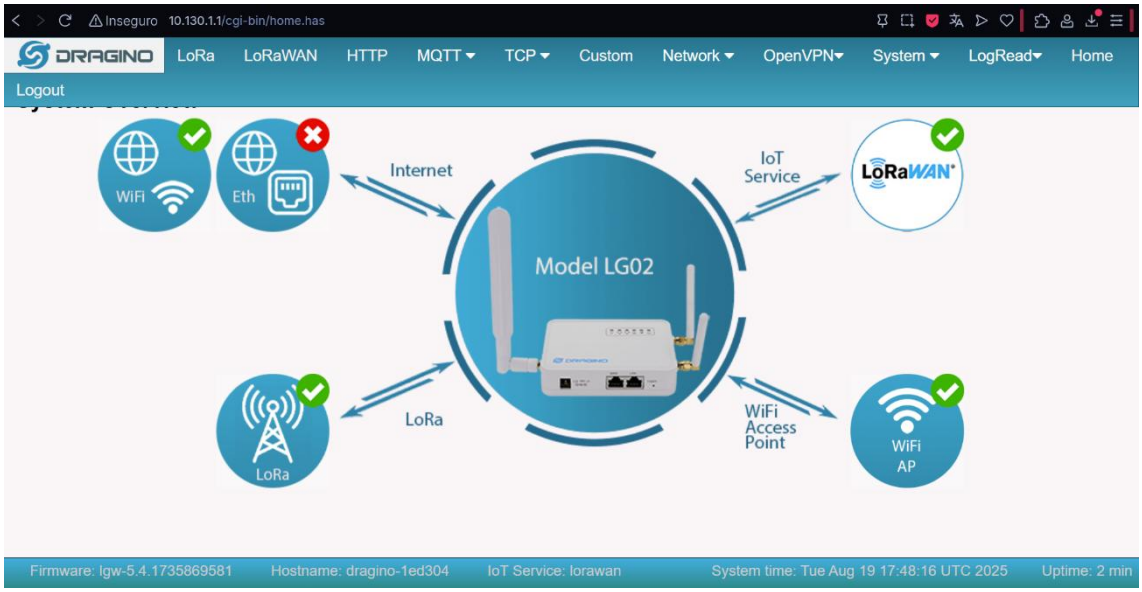

Figure A-0-1 - Home Page of the Dragino Gateway

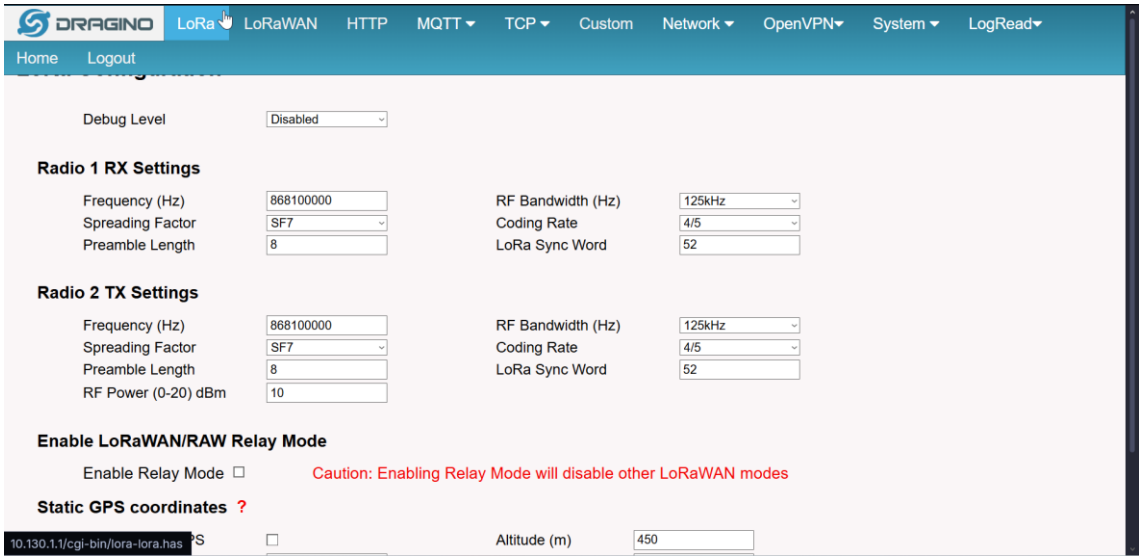

Figure A-0-2 - LoRa Page in the Dragino Gateway

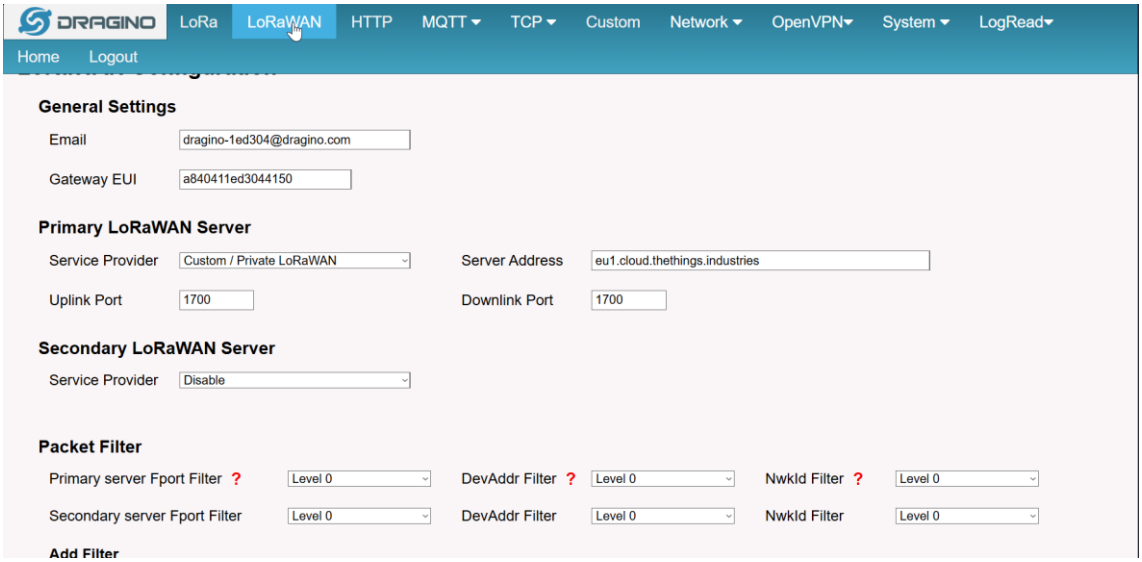

Figure A-0-3 - LoRaWAN page inside Dragino gateway

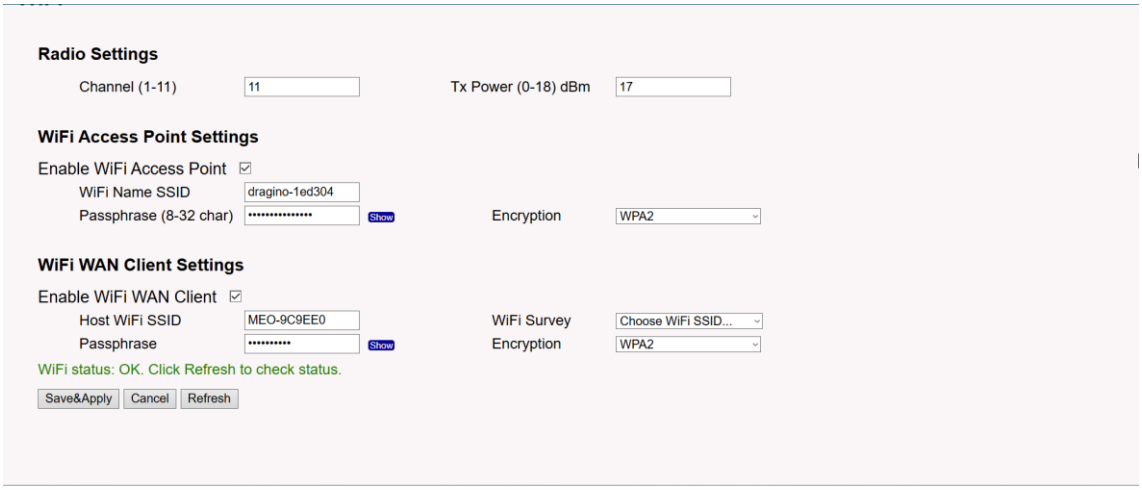

Figure A-0-4 - Configuration of the WiFi WAN access point

Figure A-0-5 - TTN LoRa Cloud server included in the gateway

ANNEX B – LoRa Cloud Configuration

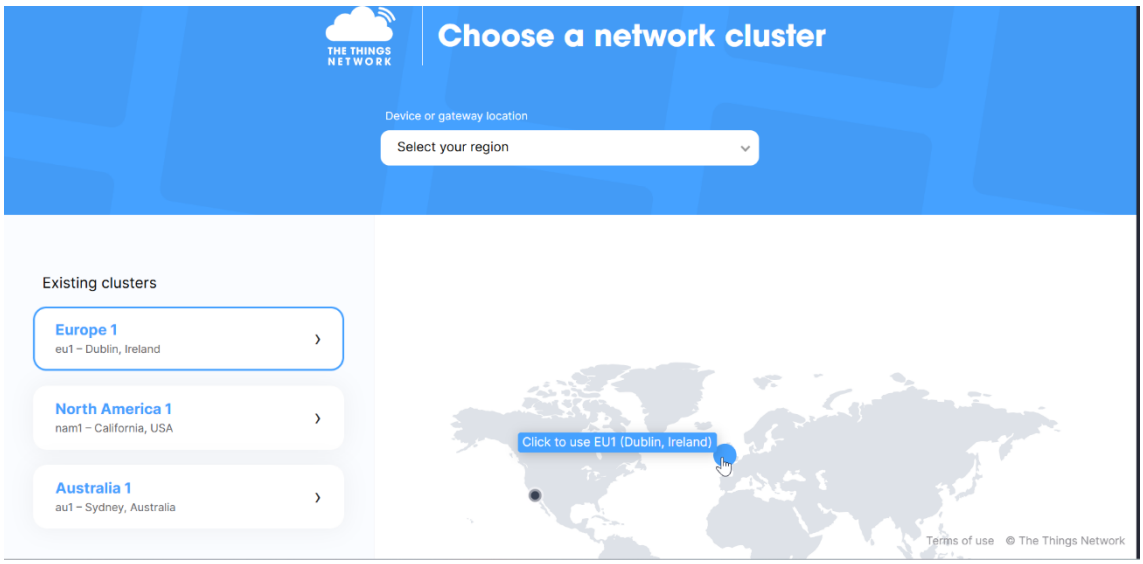

Figure B-1 - Choosing network server

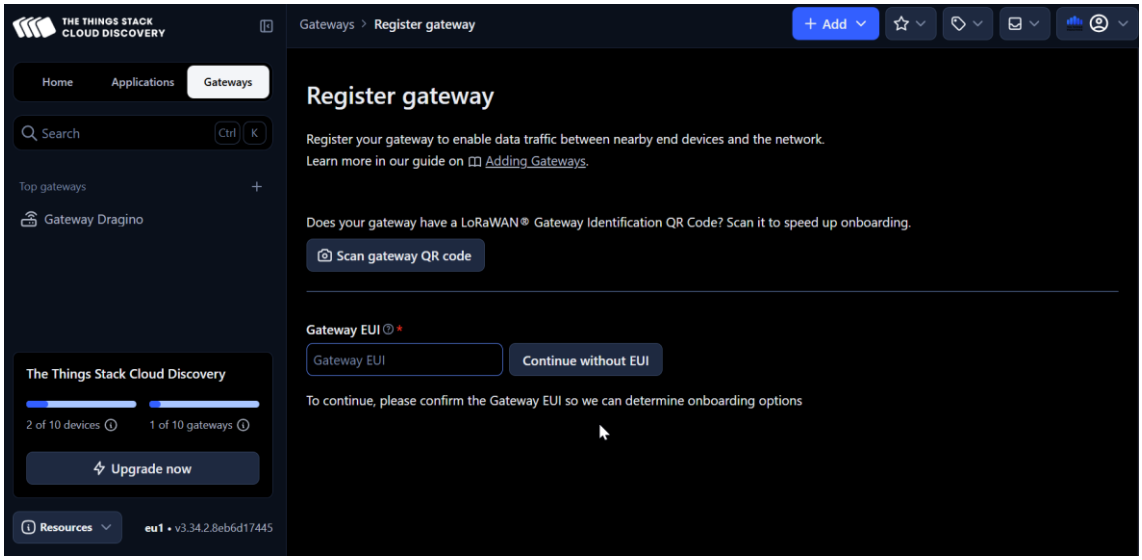

Figure B-2 - Creating the Gateway Image inside TTN LoRa Cloud

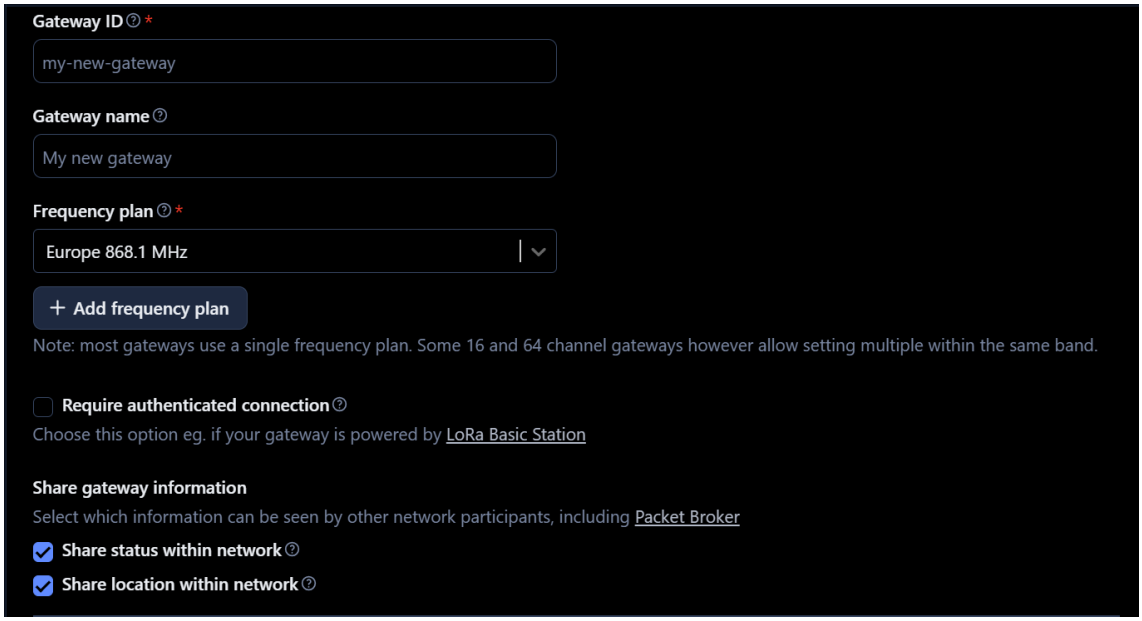

Figure B-3 - The rest of the Gateway's configuration

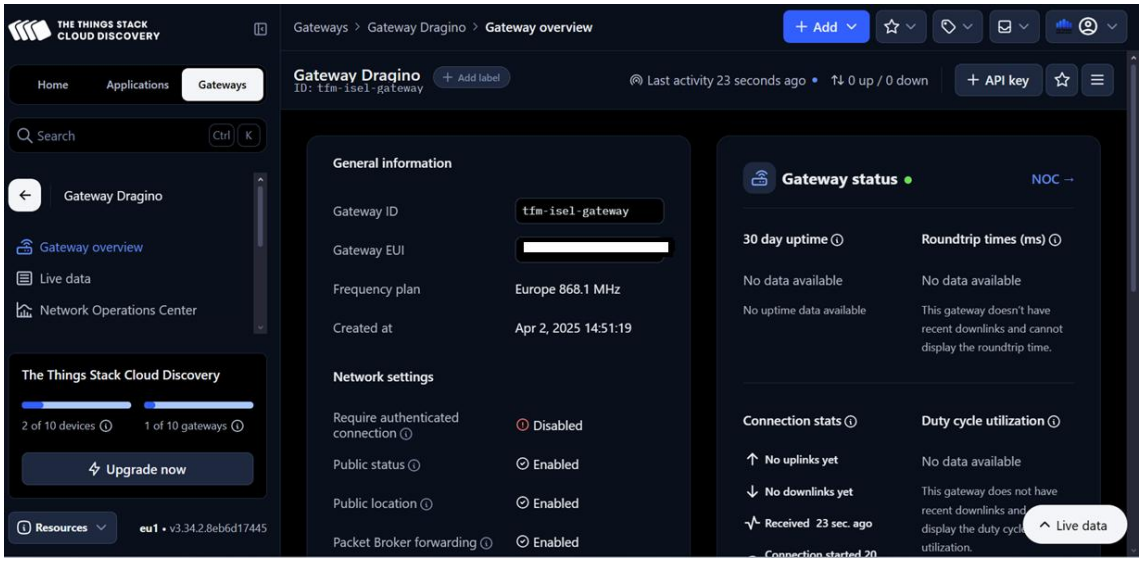

Figure B-4 - Gateway configured inside the TTN Cloud

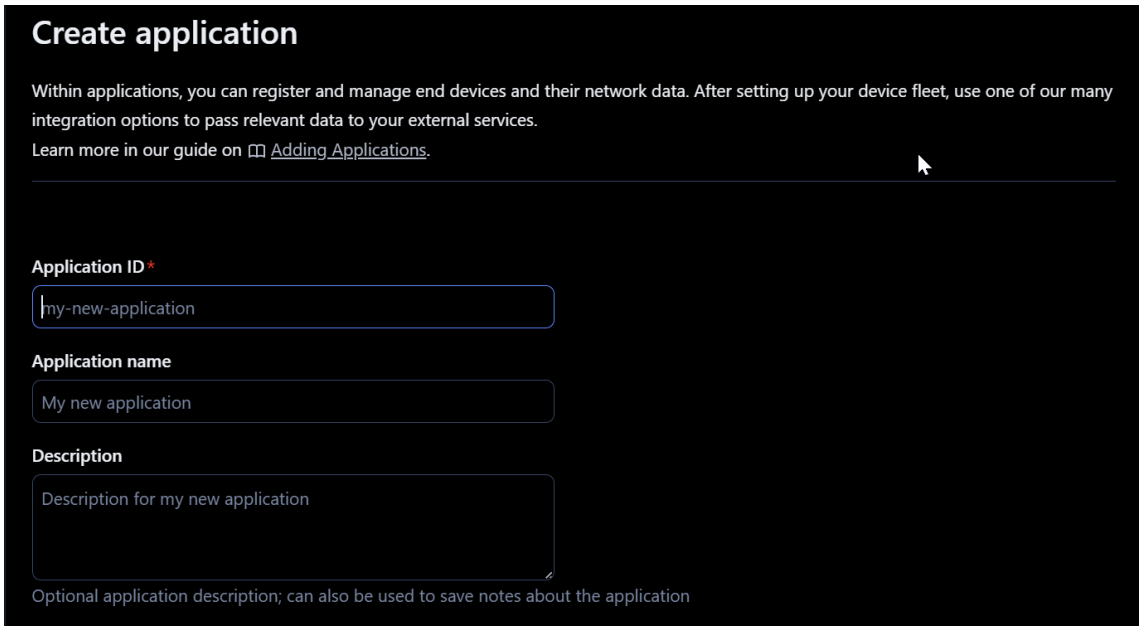

Figure B-5 - Creating a new application

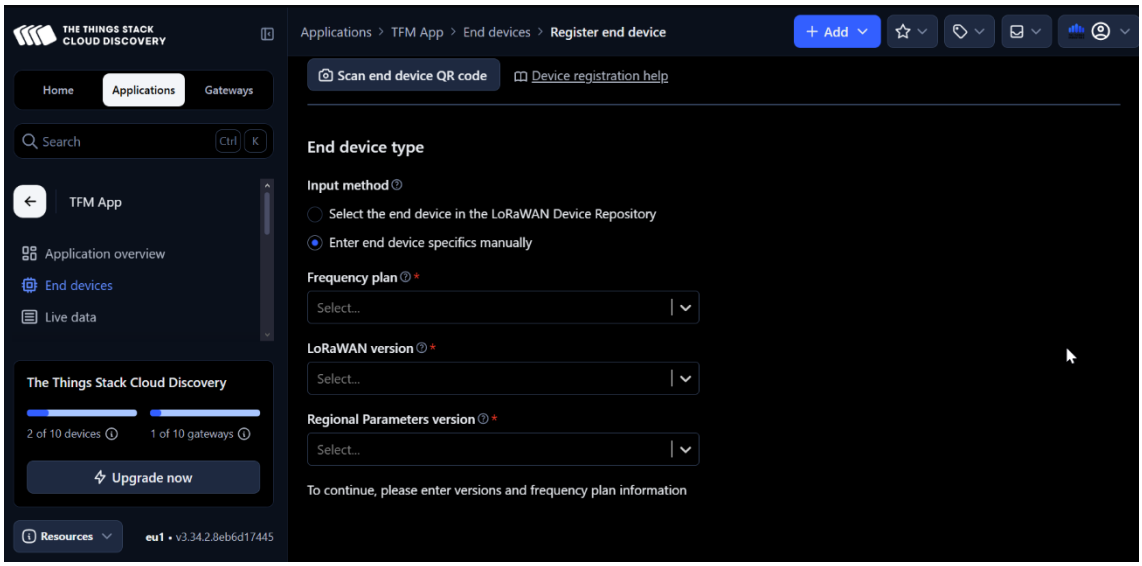

Figure B-6 - Configuration of the end device

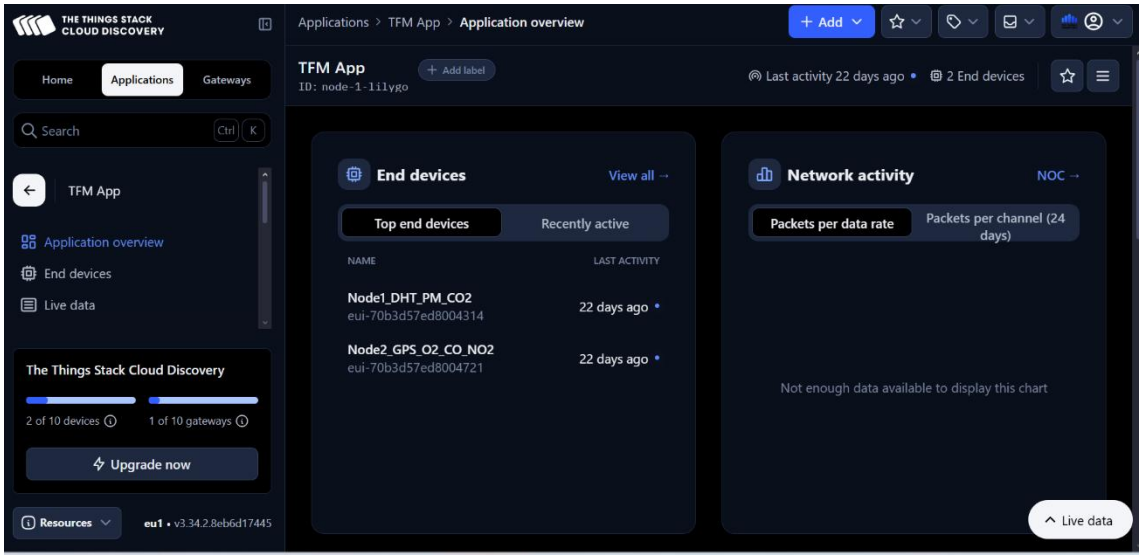

Figure B-7 - Application with both end devices

The following code is the payload formatter for Node 1:

```
function decodeUplink(input) {
  if (input.bytes.length < 10) {
    return {
      errors: ["Payload demasiado curto"],
    };
  }
  const tempRaw = ((input.bytes[0] << 8) | input.bytes[1]);
  const humRaw = ((input.bytes[2] << 8) | input.bytes[3]);
  const pm25 = ((input.bytes[4] << 8) | input.bytes[5]);
  const pm10 = ((input.bytes[6] << 8) | input.bytes[7]);
  const co2_ppm = ((input.bytes[8] << 8) | input.bytes[9]);
  return {
    data: {
      temperature: tempRaw / 100,
      humidity: humRaw / 100,
      pm2_5: pm25,
      pm10: pm10,
      co2: co2_ppm
    }
  };
}
```

The following code is the payload format for Node 2:

```
function decodeUplink(input) {
  const bytes = input.bytes;
  if (bytes.length < 14) {
    return {
      errors: ["Payload too short"],
```

```
    };
  }
  // Latitude (4 bytes - signed int32)
  const latRaw = (bytes[0] << 24) | (bytes[1] << 16) | (bytes[2] << 8)
| bytes[3];
  const lat = latRaw > 0x7FFFFFFF ? latRaw - 0x100000000 : latRaw;
  // Longitude (4 bytes - signed int32)
  const lonRaw = (bytes[4] << 24) | (bytes[5] << 16) | (bytes[6] << 8)
| bytes[7];
  const lon = lonRaw > 0x7FFFFFFF ? lonRaw - 0x100000000 : lonRaw;
  // Byte 8 é reservado/ignorado
  const o2_percent = bytes[9]; // Byte 9
  const no2_ppm = ((bytes[10] << 8) | bytes[11]) / 1000.0; // Bytes 10-
11
  const co_raw = (bytes[12] << 8) | bytes[13]; // Bytes 12-13
  return {
    data: {
      o2: o2_percent,
      no2: no2_ppm,
      co: co_raw
    }
  };
}
```

ANNEX C – Source Code

The following code belongs to Node 1 inside the MCU:

```
/**
 * @file      loramac.cpp
 * LMIC library only support SX1276 Radio
 */

#include <Arduino.h>
#include <lmic.h>
#include <hal/hal.h>
#include <TinyGPS++.h>
#include <HardwareSerial.h>
#include <DHT.h>
#include <DHT_U.h>
#include <Tomoto_HM330X.h>
#include <Wire.h>
#include "PMsystem.h"
#include "PMplatform.h"
#include "PMutility.h"
#include "LoRaBoards.h"
#include "utilities.h"


// LoRaWAN NwkSKey, network session key
static const PROGMEM u1_t NWKSKEY[16] = { 0x50, 0xA3, 0xF3, 0x39, 0xC5,
0x33, 0x53, 0x4D, 0x30, 0x24, 0xEA, 0x53, 0x06, 0xBA, 0x68, 0xAE };
// LoRaWAN AppSKey, application session key
static const PROGMEM u1_t APPSKEY[16] = { 0xCE, 0xB6, 0x14, 0x9A, 0x60,
0xE3, 0x70, 0xB1, 0x18, 0x26, 0x19, 0x70, 0xCD, 0x4D, 0xF1, 0x2B };
// LoRaWAN end-device address (DevAddr)
static const u4_t DEVADDR = 0x27FC038A;

// Pin mapping
#ifdef STM32L073xx
const lmic_pinmap lmic_pins = {
    .nss =  RADIO_CS_PIN,
    .rxtx = RADIO_SWITCH_PIN,
    .rst =  RADIO_RST_PIN,
    .dio = {RADIO_DIO0_PIN, RADIO_DIO1_PIN, RADIO_DIO2_PIN},
    .rx_level = HIGH
};
```

```
#else
const lmic_pinmap lmic_pins = {
    .nss =  RADIO_CS_PIN,
    .rxtx = LMIC_UNUSED_PIN,
    .rst =  RADIO_RST_PIN,
    .dio = {RADIO_DIO0_PIN, RADIO_DIO1_PIN, RADIO_DIO2_PIN}
};
#endif

static osjob_t sendjob;
unsigned long lastGpsSendTime = 0;
const unsigned long intervaloGps = 120000; // 2 minutos
static int spreadFactor = DR_SF7;
static int joinStatus = EV_JOINING;
static const unsigned TX_INTERVAL = 30;
static String lora_msg = "";

// ----------------------------------- GPS e Sensores -----------------
---------------------------------------
// --------------------------- GPS ---------------------------
TinyGPSPlus gps;
HardwareSerial hs(1); // UART#1 do LilyGo

// ---------------------- Hum e Temp ------------------------
#define DHTPIN  25
#define DHTTYPE DHT22
DHT_Unified dht(DHTPIN, DHTTYPE);

// -------------------- PM2.5 e PM 10 ----------------------
Tomoto_HM330X sensor;
uint64_t failureStartTime = 0;
const uint64_t RESTART_TIMEOUT = 20000;
void initSystem() {}
void haltSystem() {while (true);}
void restartSystem () {}

// -------------------- CO2 ----------------------
#define MG_PIN 35
#define BOOL_PIN 2
#define DC_GAIN 8.5
#define READ_SAMPLE_INTERVAL 50
#define READ_SAMPLE_TIMES 5
```

```
#define ZERO_POINT_VOLTAGE 0.346 //resultante de medição de 2.94 V / 8.5
V --> DC Gain
#define REACTION_VOLTAGE 0.030 //valor com atmosfera em 1000 ppm (Valor
testado em laboratório)
float CO2Curve[3] = {2.602,ZERO_POINT_VOLTAGE,(REACTION_VOLTAGE/(2.602-
3))};

// ---------------------------------- OTAA - Não necessário ----------
----------------------------------------
void os_getArtEui (u1_t *buf){}
void os_getDevEui (u1_t *buf){}
void os_getDevKey (u1_t *buf){}
// ------------------------------------------------------------------------
----------------------------------------

float MGRead(int mg_pin)
{
    int i;
    float v=0;

    for (i=0;i<READ_SAMPLE_TIMES;i++) {
        v += analogRead(mg_pin);
        delay(READ_SAMPLE_INTERVAL);
    }
    v = (v/READ_SAMPLE_TIMES) * 3.3/4095.0 ;
    return v;
}

float MGGetPercentage(float volts, float *pcurve){
    if ((volts/DC_GAIN )>=ZERO_POINT_VOLTAGE) {
      return -1;
   } else {
      return pow(10, ((volts/DC_GAIN)-pcurve[1])/pcurve[2]+pcurve[0]);
   }
}

bool my_payload(uint8_t *payload) {
    // Leitura GPS
    int32_t lat = gps.location.lat() * 1e6;
    int32_t lng = gps.location.lng() * 1e6;
    uint8_t sats = gps.satellites.value();

    // Leitura DHT
```

```
    sensors_event_t tempEvent, humEvent;
    dht.temperature().getEvent(&tempEvent);
    dht.humidity().getEvent(&humEvent);

    if                  (isnan(tempEvent.temperature)                  ||
isnan(humEvent.relative_humidity)) {
        Serial.println("Erro ao ler DHT22");
        return false;
    }

    int16_t temp_int = round(tempEvent.temperature); // décimos de °C
    int16_t hum_int  = round(humEvent.relative_humidity);    // décimos
de %

    //Leitura HM3301
    uint16_t pm10 = sensor.atm.getPM10();
    uint16_t pm2_5 = sensor.atm.getPM2_5();

    //Leitura CO2
    float volts, percentage;
    //int percentage;       //only after 48 hours
    volts = MGRead(MG_PIN);
    percentage = volts*(-400/2.94)+800;
    uint16_t co2_ppm = percentage;

   /*/ // Debug
    Serial.print("Lat: "); Serial.println(gps.location.lat(), 6);
    Serial.print("Lng: "); Serial.println(gps.location.lng(), 6);
    Serial.print("Satélites: "); Serial.println(sats);
    Serial.print("Temperatura: "); Serial.print(tempEvent.temperature);
Serial.println(" °C");
    Serial.print("Humidade:                                          ");
Serial.print(humEvent.relative_humidity); Serial.println(" %");
    Serial.print("PM2.5:    ");    Serial.print(sensor.atm.getPM2_5());
Serial.println(" ug/m^3");
    Serial.print("PM10:     ");     Serial.print(sensor.atm.getPM10());
Serial.println(" ug/m^3");
*/
    // GPS
    payload[0] = (lat >> 24) & 0xFF;
    payload[1] = (lat >> 16) & 0xFF;
    payload[2] = (lat >> 8) & 0xFF;
    payload[3] = lat & 0xFF;
```

```
    payload[4] = (lng >> 24) & 0xFF;
    payload[5] = (lng >> 16) & 0xFF;
    payload[6] = (lng >> 8) & 0xFF;
    payload[7] = lng & 0xFF;
    payload[8] = sats;

    // Temperatura (2 bytes, int16)
    payload[9]  = (temp_int >> 8) & 0xFF;
    payload[10] = temp_int & 0xFF;

    // Humidade (2 bytes, int16)
    payload[11] = (hum_int >> 8) & 0xFF;
    payload[12] = hum_int & 0xFF;

    //PM2.5 (2 bytes, int16)
    payload[13] = (pm2_5 >> 8) & 0xFF;
    payload[14] = pm2_5 & 0xFF;

    //PM10 (2 bytes, int16)
    payload[15] = (pm10 >> 8) & 0xFF;
    payload[16] = pm10 & 0xFF;

    //CO2 (2 bytes, int16)
    payload[17] = (co2_ppm >> 8) & 0xFF;
    payload[18] = co2_ppm & 0xFF;
    return true;
}

void send_payload() {
    // Verificar se dados do GPS são válidos
    if (!gps.location.isValid()) {
        Serial.println("Sem fix GPS");
        return;
    }

    // Gerar payload combinado (GPS + DHT + PM2.5 + PM10)
    uint8_t payload[19];
    if (!my_payload(payload)) {
        Serial.println("Erro ao gerar payload");
    return;
    }

    // Enviar via LoRa
```

```
    LMIC_setTxData2(1, payload, sizeof(payload), 0);
    Serial.println("Payload GPS + DHT + Sensores enviado!");
}

void send_onlySensors_payload() {
    sensors_event_t tempEvent, humEvent;
    dht.temperature().getEvent(&tempEvent);
    dht.humidity().getEvent(&humEvent);

    if (isnan(tempEvent.temperature) ||
isnan(humEvent.relative_humidity)) {
        Serial.println("Erro ao ler DHT22");
        return;
    }

    int16_t temp_int = round(tempEvent.temperature * 100);
    int16_t hum_int = round(humEvent.relative_humidity * 100);
    //Leitura HM3301
    uint16_t pm10 = sensor.atm.getPM10();
    uint16_t pm2_5 = sensor.atm.getPM2_5();
    //Leitura CO2
    float volts, percentage;
    //int percentage;       //only after 48 hours
    volts = MGRead(MG_PIN);
    percentage = volts*(-400/2.94)+800;
    uint16_t co2_ppm = percentage;

    uint8_t payload[10];
    payload[0] = (temp_int >> 8) & 0xFF;
    payload[1] = temp_int & 0xFF;
    payload[2] = (hum_int >> 8) & 0xFF;
    payload[3] = hum_int & 0xFF;
    payload[4] = (pm2_5 >> 8) & 0xFF;
    payload[5] = pm2_5 & 0xFF;
    payload[6] = (pm10 >> 8) & 0xFF;
    payload[7] = pm10 & 0xFF;
    payload[8] =(co2_ppm >> 8) & 0xFF;
    payload[9] = co2_ppm & 0xFF;

    LMIC_setTxData2(1, payload, sizeof(payload), 0);
    Serial.println("Payload Sensores enviado!");
    Serial.print("TEMP: ");
    Serial.println(temp_int);
```

```
    Serial.print("HUM: ");
    Serial.println(hum_int);
}


/*
void send_gps_payload() {
    if (!gps.location.isValid()) {
        Serial.println("Sem fix GPS");
        return;
    }
    int32_t lat = gps.location.lat() * 1e6;
    int32_t lng = gps.location.lng() * 1e6;
    uint8_t sats = gps.satellites.value();

    uint8_t payload[9];
    payload[0] = (lat >> 24) & 0xFF;
    payload[1] = (lat >> 16) & 0xFF;
    payload[2] = (lat >> 8) & 0xFF;
    payload[3] = lat & 0xFF;
    payload[4] = (lng >> 24) & 0xFF;
    payload[5] = (lng >> 16) & 0xFF;
    payload[6] = (lng >> 8) & 0xFF;
    payload[7] = lng & 0xFF;
    payload[8] = sats;

    LMIC_setTxData2(1, payload, sizeof(payload), 0);
    Serial.println("Payload GPS enviado!");
}
*/

void do_send(osjob_t *j)
{
    if (LMIC.opmode & OP_TXRXPEND) {
        Serial.println(F("OP_TXRXPEND, not sending"));
        return;
    }

    Serial.println(F("OP_TXRXPEND, sending ..."));

    unsigned long now = millis();
```

```
    if ((now - lastGpsSendTime) >= intervaloGps || lastGpsSendTime ==
0) {
        if (gps.location.isValid()) {
            send_payload(); // envia GPS + DHT
            lastGpsSendTime = now;
        } else {
            Serial.println(F("GPS inválido, enviando só sensores"));
            send_onlySensors_payload();
        }
    } else {
        send_onlySensors_payload(); // envia só DHT durante os 2 minutos
    }

    if (u8g2) {
        char buf[256];
        u8g2->clearBuffer();
        snprintf(buf,  sizeof(buf),  "[%lu]data  sending!",  millis()  /
1000);
        u8g2->drawStr(0, 12, buf);
        u8g2->sendBuffer();
    }
}
/*
-------------------------------- DO SEND SÓ LoRa ----------------------
---------------------------------------
void do_send(osjob_t *j)
{
   /* if (joinStatus == EV_JOINING) {
        Serial.println(F("Not joined yet"));
        // Check if there is not a current TX/RX job running
        os_setTimedCallback(&sendjob,          os_getTime()          +
sec2osticks(TX_INTERVAL), do_send);

    } else if (LMIC.opmode & OP_TXRXPEND) {
        Serial.println(F("OP_TXRXPEND, not sending"));
    } else {
        Serial.println(F("OP_TXRXPEND,sending ..."));
        static uint8_t mydata[] = "Hello, world!";
        // Prepare upstream data transmission at the next possible time.
        LMIC_setTxData2(1, mydata, sizeof(mydata) - 1, 0);
        //os_setTimedCallback(&sendjob,          os_getTime()          +
sec2osticks(TX_INTERVAL), do_send);
        if (u8g2) {
```

```
            char buf[256];
            u8g2->clearBuffer();
            snprintf(buf, sizeof(buf), "[%lu]data sending!", millis() /
1000);
            u8g2->drawStr(0, 12, buf);
            u8g2->sendBuffer();
        }
    }
*/

void onEvent (ev_t ev)
{
    Serial.print(os_getTime());
    Serial.print(": ");
    switch (ev) {
    case EV_TXCOMPLETE:
        Serial.println(F("EV_TXCOMPLETE (includes waiting for RX
windows)"));

        if (LMIC.txrxFlags & TXRX_ACK) {
            Serial.println(F("Received ack"));
            lora_msg =  "Received ACK.";
        }

        lora_msg = "rssi:" + String(LMIC.rssi) + " snr: " +
String(LMIC.snr);

        if (LMIC.dataLen) {
            // data received in rx slot after tx
            Serial.print(F("Data Received: "));
            // Serial.write(LMIC.frame + LMIC.dataBeg, LMIC.dataLen);
            // Serial.println();
            Serial.println(LMIC.dataLen);
            Serial.println(F(" bytes of payload"));
        }
        // Schedule next transmission
        os_setTimedCallback(&sendjob, os_getTime() +
sec2osticks(TX_INTERVAL), do_send);
        break;
    case EV_JOINING:
         Serial.println(F("EV_JOINING"));
        /*Serial.println(F("EV_JOINING: -> Joining..."));
        lora_msg = "OTAA joining....";
```

```
    joinStatus = EV_JOINING;

    if (u8g2) {
        u8g2->clearBuffer();
        u8g2->drawStr(0, 12, "OTAA joining....");
        u8g2->sendBuffer();
    }*/
    break;
case EV_JOIN_FAILED:
    Serial.println(F("EV_JOINING_FAILED"));
    /*Serial.println(F("EV_JOIN_FAILED: -> Joining failed"));
    lora_msg = "OTAA Joining failed";
    if (u8g2) {
        u8g2->clearBuffer();
        u8g2->drawStr(0, 12, "OTAA joining failed");
        u8g2->sendBuffer();
    }*/
    break;
case EV_JOINED:
    Serial.println(F("EV_JOINED"));
    /*Serial.println(F("EV_JOINED"));
    lora_msg = "Joined!";
    joinStatus = EV_JOINED;

    if (u8g2) {
        u8g2->clearBuffer();
        u8g2->drawStr(0, 12, "Joined TTN!");
        u8g2->sendBuffer();
    }
    delay(3);
    // Disable link check validation (automatically enabled
    // during join, but not supported by TTN at this time).
    LMIC_setLinkCheckMode(0);*/

    break;
case EV_RXCOMPLETE:
    // data received in ping slot
    Serial.println(F("EV_RXCOMPLETE"));
    break;
case EV_LINK_DEAD:
    Serial.println(F("EV_LINK_DEAD"));
    break;
case EV_LINK_ALIVE:
```

```
        Serial.println(F("EV_LINK_ALIVE"));
        break;
    default:
        Serial.println(F("Unknown event"));
        break;
    }
}

u1_t readReg (u1_t addr)
{
    hal_pin_nss(0);
    hal_spi(addr & 0x7F);
    u1_t val = hal_spi(0x00);
    hal_pin_nss(1);
    return val;
}

uint32_t delayMS;

void setupLMIC(void)
{

    Serial.begin(115200);

    //GPS receiver
    hs.begin(9600, SERIAL_8N1, 34, 12);
    Serial.println(F("Iniciar GPS..."));

    //DHT22 sensor
    dht.begin();
    Serial.println(F("Iniciar DHT22 sensor..."));

    //for Dust Sensor
    initSystem();
    delay(100);
    if (!sensor.begin()) {
        Serial.println("Failed to initialize the sensor.");
        haltSystem();
    }
    Serial.println("Sensor initialized.");

    //for CO2
    pinMode(BOOL_PIN, INPUT);
```

```
    digitalWrite(BOOL_PIN, HIGH);

    //setupLMIC();
    #ifdef  RADIO_TCXO_ENABLE
        pinMode(RADIO_TCXO_ENABLE, OUTPUT);
        digitalWrite(RADIO_TCXO_ENABLE, HIGH);
    #endif

    // LMIC init
    os_init();

    // Reset the MAC state. Session and pending data transfers will be
discarded.
    LMIC_reset();

    LMIC_setClockError(MAX_CLOCK_ERROR * 1 / 100);
    // Set up the channels used by the Things Network, which corresponds
    // to the defaults of most gateways. Without this, only three base
    // channels from the LoRaWAN specification are used, which certainly
    // works, so it is good for debugging, but can overload those
    // frequencies, so be sure to configure the full frequency range of
    // your network here (unless your network autoconfigures them).
    // Setting up channels should happen after LMIC_setSession, as that
    // configures the minimal channel set.

    uint8_t appskey[sizeof(APPSKEY)];
    uint8_t nwkskey[sizeof(NWKSKEY)];
    memcpy_P(appskey, APPSKEY, sizeof(APPSKEY));
    memcpy_P(nwkskey, NWKSKEY, sizeof(NWKSKEY));
    LMIC_setSession (0x13, DEVADDR, nwkskey, appskey);

    LMIC_setupChannel(0,   868100000,   DR_RANGE_MAP(DR_SF12,   DR_SF7),
BAND_CENTI);      // g-band
    /*LMIC_setupChannel(1,  868300000,  DR_RANGE_MAP(DR_SF12,  DR_SF7B),
BAND_CENTI);      // g-band
    LMIC_setupChannel(2,   868500000,   DR_RANGE_MAP(DR_SF12,   DR_SF7),
BAND_CENTI);      // g-band
    LMIC_setupChannel(3,   867100000,   DR_RANGE_MAP(DR_SF12,   DR_SF7),
BAND_CENTI);      // g-band
    LMIC_setupChannel(4,   867300000,   DR_RANGE_MAP(DR_SF12,   DR_SF7),
BAND_CENTI);      // g-band
    LMIC_setupChannel(5,   867500000,   DR_RANGE_MAP(DR_SF12,   DR_SF7),
BAND_CENTI);      // g-band
```

```
    LMIC_setupChannel(6,   867700000,   DR_RANGE_MAP(DR_SF12,   DR_SF7),
BAND_CENTI);      // g-band
    LMIC_setupChannel(7,   867900000,   DR_RANGE_MAP(DR_SF12,   DR_SF7),
BAND_CENTI);      // g-band
    LMIC_setupChannel(8,   868800000,   DR_RANGE_MAP(DR_FSK,    DR_FSK),
BAND_MILLI); */     // g2-band
    // TTN defines an additional channel at 869.525Mhz using SF9 for
class B
    // devices' ping slots. LMIC does not have an easy way to define set
this
    // frequency and support for class B is spotty and untested, so this
    // frequency is not configured here.

    // Disable link check validation
    LMIC_setLinkCheckMode(0);

    // TTN uses SF9 for its RX2 window.
    LMIC.dn2Dr = DR_SF9;

    // Set data rate and transmit power for uplink (note: txpow seems
to be ignored by the library)
    LMIC_setAdrMode(0);
    LMIC_setDrTxpow(DR_SF7, 14);

    /*Serial.println("LMIC_startJoining");
    // Start job
    LMIC_startJoining();*/
    sensor_t sensor;
    delayMS = sensor.min_delay/1000;

    do_send(&sendjob);     // Will fire up also the join
}

void loopLMIC(void)
{
    while (hs.available()) {
        char c = hs.read();
        Serial.write(c);  // mostra dados brutos
        gps.encode(c);
    }

    if (gps.location.isUpdated()) {
        Serial.print("Lat: "); Serial.println(gps.location.lat(), 6);
```

```
        Serial.print("Lng: "); Serial.println(gps.location.lng(), 6);
    }
    if (millis() > 15000 && gps.charsProcessed() < 10) {
        Serial.println(F("No GPS detected: check wiring."));
        delay(15000);
    }

    State sensorState;
    if (!sensor.readSensor()) {
        Serial.println("Failed to read the sensor.");
        sensorState = State::BAD;

    if (failureStartTime) {
        // Restart the system if the sensor has not responded for a
while
        if (millis() - failureStartTime > RESTART_TIMEOUT) {
        Serial.println("Sensor is not responding. Restarting...");
        restartSystem();
        }
    } else {
        failureStartTime = millis();  // Failure started
    }
    } else {
    sensorState = millis() < 30 * 1000
                        ? State::WARN   // Sensor needs 30 seconds to
become stable
                        : State::GOOD;

    failureStartTime = 0;  // Clear failure
    }
    os_runloop_once();
```

The following code belongs to node 2 inside the MCU:

```
/**
 * @file      node2_loramac.cpp
 * LMIC library only support SX1276 Radio
 */

#include <Arduino.h>
#include <lmic.h>
#include <hal/hal.h>
#include <TinyGPS++.h>
```

```
#include <HardwareSerial.h>
#include "DFRobot_MultiGasSensor.h"
#include "LoRaBoards.h"
#include "utilities.h"

// LoRaWAN NwkSKey, network session key
// This should be in big-endian (aka msb).
static const PROGMEM u1_t NWKSKEY[16] = { 0xB0, 0x4C, 0x37, 0x11, 0x57,
0xC2, 0x7F, 0xF5, 0x6A, 0x7B, 0xD8, 0x90, 0xCD, 0x7E, 0x0E, 0xAB };
// LoRaWAN AppSKey, application session key
// This should also be in big-endian (aka msb).
static const u1_t PROGMEM APPSKEY[16] = { 0x36, 0xEF, 0xC6, 0x91, 0x35,
0xD3, 0x21, 0xA5, 0x85, 0xF2, 0x04, 0xCE, 0x94, 0x1E, 0x9D, 0x6B };
// LoRaWAN end-device address (DevAddr)
static const u4_t DEVADDR = 0x27FC038D;

// Pin mapping
#ifdef STM32L073xx
const lmic_pinmap lmic_pins = {
    .nss =  RADIO_CS_PIN,
    .rxtx = RADIO_SWITCH_PIN,
    .rst =  RADIO_RST_PIN,
    .dio = {RADIO_DIO0_PIN, RADIO_DIO1_PIN, RADIO_DIO2_PIN},
    .rx_level = HIGH
};
#else
const lmic_pinmap lmic_pins = {
    .nss =  RADIO_CS_PIN,
    .rxtx = LMIC_UNUSED_PIN,
    .rst =  RADIO_RST_PIN,
    .dio = {RADIO_DIO0_PIN, RADIO_DIO1_PIN, RADIO_DIO2_PIN}
};
#endif

static osjob_t sendjob;
static int spreadFactor = DR_SF7;
static int joinStatus = EV_JOINING;
static const unsigned TX_INTERVAL = 30;
static String lora_msg = "";
//Potência de 6dBm para ser o melhor valor possível

// ---------------------------------- GPS e Sensores -----------------
----------------------------------------
```

```
// -------------------------- GPS --------------------------
TinyGPSPlus gps;
HardwareSerial hs(1);

// -------------------------- O2 ---------------------------
#define I2C_COMMUNICATION

#ifdef  I2C_COMMUNICATION
#define I2C_ADDRESS    0x74
DFRobot_GAS_I2C gas(&Wire ,I2C_ADDRESS);
#else
#if (!defined ARDUINO_ESP32_DEV) && (!defined __SAMD21G18A__)
/**
  UNO:pin_2-----RX
      pin_3-----TX
*/
  SoftwareSerial mySerial(2,3);
  DFRobot_GAS_SoftWareUart gas(&mySerial);
#else
/**
  ESP32:IO16-----RX
        IO17-----TX
*/
  DFRobot_GAS_HardWareUart gas(&Serial2); //ESP32HardwareSerial
#endif
#endif

// -------------------------- NO2 --------------------------
int No2_pin = 35;

// -------------------------- CO ---------------------------
int CO_pin = 32;

// ---------------------------------- OTAA - Não necessário ----------
----------------------------------------
void os_getArtEui (u1_t *buf){}
void os_getDevEui (u1_t *buf){}
void os_getDevKey (u1_t *buf){}
// ------------------------------------------------------------------------
----------------------------------------

static uint8_t Payload[14];
//static uint8_t Payload[8];
```

```
/*void do_send(osjob_t *j)
{
    if (LMIC.opmode & OP_TXRXPEND) {
        Serial.println(F("OP_TXRXPEND, not sending"));
    } else {
        Serial.println(F("Sending GPS data..."));

        // Atualiza leitura do GPS
        while (Serial1.available() > 0) {
            gps.encode(Serial1.read());
        }

        if (gps.location.isValid()) {
            // Converte lat/lon em inteiros para compactar no payload
(exemplo: graus * 10000)
            int32_t lat = (int32_t)(gps.location.lat() * 10000);
            int32_t lon = (int32_t)(gps.location.lng() * 10000);
            int16_t alt = (int16_t)gps.altitude.meters();

            // Coloca latitude (3 bytes) no payload (por simplicidade
aqui usamos 4 bytes, mas pode ser 3 bytes com shift)
            Payload[0] = (lat >> 24) & 0xFF;
            Payload[1] = (lat >> 16) & 0xFF;
            Payload[2] = (lat >> 8) & 0xFF;
            Payload[3] = lat & 0xFF;

            // Longitude (4 bytes)
            Payload[4] = (lon >> 24) & 0xFF;
            Payload[5] = (lon >> 16) & 0xFF;
            Payload[6] = (lon >> 8) & 0xFF;
            Payload[7] = lon & 0xFF;

            //LMIC_setTxData2(1, Payload, 8, 0);
            //Serial.printf("Enviando    GPS:    lat=%.5f    lon=%.5f
alt=%.1f\n", gps.location.lat(), gps.location.lng());

        } else {
            Serial.println(F("GPS sem fix"));
            // Envia um payload vazio ou outro dado qualquer enquanto
não tem fix
            //uint8_t noFixPayload[] = {0};
```

```
            //LMIC_setTxData2(1,  noFixPayload,  sizeof(noFixPayload),
0);
        }

        uint8_t o2_con = gas.readGasConcentrationPPM();
        Payload[8] = (o2_con >> 8) & 0xFF;
        Payload[9] = o2_con & 0xFF;

        uint8_t no2_value = 0;
        no2_value = analogRead(No2_pin);
        Payload[10] = (no2_value >> 8) & 0xFF;
        Payload[11] = no2_value & 0xFF;

        Serial.print("Ambient ");
        Serial.print(gas.queryGasType());
        Serial.print(" concentration is: ");
        Serial.print(gas.readGasConcentrationPPM());
        Serial.println(" %vol");
        Serial.print("No2 value: ");
        Serial.print(no2_value + "ppm");
        LMIC_setTxData2(1, Payload, sizeof(Payload), 0);
        Serial.printf("Enviando  GPS:  lat=%.5f  lon=%.5f  alt=%.1f\n",
gps.location.lat(), gps.location.lng());

    }
}*/

void do_send(osjob_t *j)
{
    if (LMIC.opmode & OP_TXRXPEND) {
        Serial.println(F("OP_TXRXPEND, not sending"));
        return;
    }

    Serial.println(F("Preparing data..."));

    // Atualiza GPS
    while (Serial1.available() > 0) {
        gps.encode(Serial1.read());
    }

    bool hasFix = gps.location.isValid();
    int32_t lat = 0;
```

```
    int32_t lon = 0;

    if (hasFix) {
        lat = gps.location.lat() * 10000;
        lon = gps.location.lng() * 10000;
    }

    // Preenche payload com zeros se não houver fix
    Payload[0] = (lat >> 24) & 0xFF;
    Payload[1] = (lat >> 16) & 0xFF;
    Payload[2] = (lat >> 8) & 0xFF;
    Payload[3] = lat & 0xFF;

    Payload[4] = (lon >> 24) & 0xFF;
    Payload[5] = (lon >> 16) & 0xFF;
    Payload[6] = (lon >> 8) & 0xFF;
    Payload[7] = lon & 0xFF;
////////////////////////////////////////////////////////////////////
/

    // Leitura do O2
    uint8_t o2_con = 0;
    if (gas.begin()) {
        o2_con = gas.readGasConcentrationPPM();
    }
    Payload[8] = 0;
    Payload[9] = o2_con;
    //Payload[8]  = (o2_con >> 8) & 0xFF; // parte alta
    //Payload[9] = o2_con & 0xFF;

    // Leitura do NO2
    int no2_raw = analogRead(No2_pin); //leitura RAW
    float Vadc = no2_raw * (5 / 4095.0);  // para ESP32
    // Divisor de tensão:
    float R1 = 10000.0;
    float R2 = 6800.0;
    float Vsensor = Vadc * (R1 + R2) / R2;
    // Conversão para ppm (DFRobot SEN0574)
    float ppm = (Vsensor - 0.1) / 100;
    uint16_t No2_payload = (uint16_t)(ppm*1000);

    Payload[10] = (No2_payload >> 8) & 0xFF;
    Payload[11] = No2_payload & 0xFF;
```

```
    //Leitura CO
    int co_raw = analogRead(CO_pin);
    float co_adc = co_raw * (5/ 4095.0);
    float CO_Vsensor = co_adc * (R1 + R2) / R2;
    float CO_ppm = (Vsensor - 0.1) / 100;
    uint16_t CO_payload = (uint16_t)(CO_ppm*1000);
    //uint16_t CO_payload = co_raw;

    Payload[12] = (CO_payload >> 8) & 0xFF;
    Payload[13] = CO_payload & 0xFF;

    // Verificação final antes de enviar
    Serial.print(F("Enviando payload: "));
    /*for (int i = 0; i < sizeof(Payload); i++) {
        Serial.print(Payload[i], HEX);
        Serial.println(" ");
    }*/
    Serial.print("Ambient ");
    Serial.print(gas.queryGasType());
    Serial.print(" concentration is: ");
    Serial.print(gas.readGasConcentrationPPM());
    Serial.println(" %vol");
    Serial.print("No2 value: ");
    Serial.print(ppm);
    Serial.print(" ppm");
    Serial.println();
    Serial.print("CO value: ");
    Serial.print(co_raw);
    Serial.print(" ppm");
    Serial.println();

    LMIC_setTxData2(1, Payload, sizeof(Payload), 0);
}

void onEvent (ev_t ev)
{
    Serial.print(os_getTime());
    Serial.print(": ");
    switch (ev) {
    case EV_TXCOMPLETE:
        Serial.println(F("EV_TXCOMPLETE (includes waiting for RX
windows)"));
```

```
        if (LMIC.txrxFlags & TXRX_ACK) {
            Serial.println(F("Received ack"));
            lora_msg =  "Received ACK.";
        }

        lora_msg  =  "rssi:"  +  String(LMIC.rssi)  +  "  snr:  "  +
String(LMIC.snr);

        if (LMIC.dataLen) {
            // data received in rx slot after tx
            Serial.print(F("Data Received: "));
            // Serial.write(LMIC.frame + LMIC.dataBeg, LMIC.dataLen);
            // Serial.println();
            Serial.println(LMIC.dataLen);
            Serial.println(F(" bytes of payload"));
        }
        // Schedule next transmission
        os_setTimedCallback(&sendjob,          os_getTime()          +
sec2osticks(TX_INTERVAL), do_send);
        break;
    case EV_JOINING:
         Serial.println(F("EV_JOINING"));
        /*Serial.println(F("EV_JOINING: -> Joining..."));
        lora_msg = "OTAA joining....";
        joinStatus = EV_JOINING;

        if (u8g2) {
            u8g2->clearBuffer();
            u8g2->drawStr(0, 12, "OTAA joining....");
            u8g2->sendBuffer();
        }*/
        break;
    case EV_JOIN_FAILED:
        Serial.println(F("EV_JOINING_FAILED"));
        /*Serial.println(F("EV_JOIN_FAILED: -> Joining failed"));
        lora_msg = "OTAA Joining failed";
        if (u8g2) {
            u8g2->clearBuffer();
            u8g2->drawStr(0, 12, "OTAA joining failed");
            u8g2->sendBuffer();
        }*/
        break;
```

```
    case EV_JOINED:
        Serial.println(F("EV_JOINED"));
        /*Serial.println(F("EV_JOINED"));
        lora_msg = "Joined!";
        joinStatus = EV_JOINED;

        if (u8g2) {
            u8g2->clearBuffer();
            u8g2->drawStr(0, 12, "Joined TTN!");
            u8g2->sendBuffer();
        }
        delay(3);
        // Disable link check validation (automatically enabled
        // during join, but not supported by TTN at this time).
        LMIC_setLinkCheckMode(0);*/

        break;
    case EV_RXCOMPLETE:
        // data received in ping slot
        Serial.println(F("EV_RXCOMPLETE"));
        break;
    case EV_LINK_DEAD:
        Serial.println(F("EV_LINK_DEAD"));
        break;
    case EV_LINK_ALIVE:
        Serial.println(F("EV_LINK_ALIVE"));
        break;
    default:
        Serial.println(F("Unknown event"));
        break;
    }
}

u1_t readReg (u1_t addr)
{
    hal_pin_nss(0);
    hal_spi(addr & 0x7F);
    u1_t val = hal_spi(0x00);
    hal_pin_nss(1);
    return val;
}

void setupLMIC(void)
```

```
{
    Serial.begin(115200);

    //GPS receiver
    hs.begin(9600, SERIAL_8N1, 34, 12);
    Serial.println(F("Iniciar GPS..."));

    //O2 Sensor
    gas.changeAcquireMode(gas.PASSIVITY);
    delay(1000);
    gas.setTempCompensation(gas.ON);

    #ifdef  RADIO_TCXO_ENABLE
        pinMode(RADIO_TCXO_ENABLE, OUTPUT);
        digitalWrite(RADIO_TCXO_ENABLE, HIGH);
    #endif

    // LMIC init
    os_init();

    // Reset the MAC state. Session and pending data transfers will be
discarded.
    LMIC_reset();

    LMIC_setClockError(MAX_CLOCK_ERROR * 1 / 100);
    // Set up the channels used by the Things Network, which corresponds
    // to the defaults of most gateways. Without this, only three base
    // channels from the LoRaWAN specification are used, which certainly
    // works, so it is good for debugging, but can overload those
    // frequencies, so be sure to configure the full frequency range of
    // your network here (unless your network autoconfigures them).
    // Setting up channels should happen after LMIC_setSession, as that
    // configures the minimal channel set.

    uint8_t appskey[sizeof(APPSKEY)];
    uint8_t nwkskey[sizeof(NWKSKEY)];
    memcpy_P(appskey, APPSKEY, sizeof(APPSKEY));
    memcpy_P(nwkskey, NWKSKEY, sizeof(NWKSKEY));
    LMIC_setSession (0x13, DEVADDR, nwkskey, appskey);

    LMIC_setupChannel(0,   868100000,   DR_RANGE_MAP(DR_SF12,   DR_SF7),
BAND_CENTI);      // g-band
```

```
    /*LMIC_setupChannel(1, 868300000, DR_RANGE_MAP(DR_SF12, DR_SF7B),
BAND_CENTI);      // g-band
    LMIC_setupChannel(2, 868500000, DR_RANGE_MAP(DR_SF12, DR_SF7),
BAND_CENTI);      // g-band
    LMIC_setupChannel(3, 867100000, DR_RANGE_MAP(DR_SF12, DR_SF7),
BAND_CENTI);      // g-band
    LMIC_setupChannel(4, 867300000, DR_RANGE_MAP(DR_SF12, DR_SF7),
BAND_CENTI);      // g-band
    LMIC_setupChannel(5, 867500000, DR_RANGE_MAP(DR_SF12, DR_SF7),
BAND_CENTI);      // g-band
    LMIC_setupChannel(6, 867700000, DR_RANGE_MAP(DR_SF12, DR_SF7),
BAND_CENTI);      // g-band
    LMIC_setupChannel(7, 867900000, DR_RANGE_MAP(DR_SF12, DR_SF7),
BAND_CENTI);      // g-band
    LMIC_setupChannel(8, 868800000, DR_RANGE_MAP(DR_FSK,  DR_FSK),
BAND_MILLI); */     // g2-band
    // TTN defines an additional channel at 869.525Mhz using SF9 for
class B
    // devices' ping slots. LMIC does not have an easy way to define set
this
    // frequency and support for class B is spotty and untested, so this
    // frequency is not configured here.

    // Disable link check validation
    LMIC_setLinkCheckMode(0);

    // TTN uses SF9 for its RX2 window.
    LMIC.dn2Dr = DR_SF9;

    // Set data rate and transmit power for uplink (note: txpow seems
to be ignored by the library)
    LMIC_setDrTxpow(spreadFactor, 14);

    /*Serial.println("LMIC_startJoining");
    // Start job
    LMIC_startJoining();*/

    do_send(&sendjob);     // Will fire up also the join
}

void loopLMIC(void)
{
    os_runloop_once();
```

```
}
```