UNIVERSITY OF THE AEGEAN
SCHOOL OF ENGINEERING

DEPARTMENT OF INFORMATION & COMMUNICATION SYSTEMS ENGINEERING

Study and implementation of air quality and noise pollution monitoring system in urban areas using LoRaWAN technology

A thesis statement submitted by

Konstantinos Stathis

Committee

Supervisor: Dimitrios Skoutas – Assistant Professor
Member: Charalabos Skianis – Professor
Member: Demosthenes Vouyioukas – Professor

Samos, 2022

Η ΤΡΙΜΕΛΗΣ ΕΠΙΤΡΟΠΗ ΔΙΔΑΣΚΟΝΤΩΝ ΕΠΙΚΥΡΩΝΕΙ
ΤΗΝ ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ
ΤΟΥ
ΚΩΝΣΤΑΝΤΙΝΟΥ ΣΤΑΘΗ

_____
Δημήτριος Σκούτας, Επίκουρος Καθηγητής

_____
Χαράλαμπος Σκιάνης, Καθηγητής

_____
Δημοσθένης Βουγιούκας, Καθηγητής

ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΙΓΑΙΟΥ
ΣΑΜΟΣ
2022

Konstantinos Stathis
Undergraduate student
Dept. of Information and Communication Systems Engineering
School of Engineering
University of the Aegean

Βεβαιώνω ότι είμαι συγγραφέας αυτής της διπλωματικής εργασίας και δηλώνω ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω αναφέρει τις πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες.

Τέλος, βεβαιώνω ότι αυτή η διπλωματική εργασία προετοιμάστηκε από εμένα προσωπικά ειδικά για τις απαιτήσεις του προπτυχιακού προγράμματος σπουδών του Τμήματος Μηχανικών Πληροφοριακών και Επικοινωνιακών Συστημάτων του Πανεπιστημίου Αιγαίου στη Σάμο.

I declare that this thesis is my own work and was written without literature other than the sources indicated in the bibliography. Information used from the published or unpublished work of others has been acknowledged in the text and has been explicitly referred to in the given list of references. This thesis has not been submitted in any form for another degree or diploma at any university or other institute of tertiary education.

Finally, I declare that this thesis is submitted in fulfilment of the requirements for the degree of Engineer of Information and Communication Systems Engineering in the University of the Aegean, of the Department of Information and Communication Systems Engineering (ICSD).

This page intentionally left blank

Πρόλογος και ευχαριστίες

Σε αυτό το σημείο μου δίνεται η ευκαιρία να εκφράσω την ευγνωμοσύνη και τις ευχαριστίες προς την οικογένειά μου, η οποία μου παρείχε έμπρακτα όσα χρειάστηκαν για να φοιτήσω και με στήριξαν σε όλη την διάρκεια των σπουδών μου. Οι γονείς μου και τα αδέλφια μου, αποτελούν έμπνευση και αρωγοί σε όλα μου τα βήματά και δεν σταματούν ποτέ να είναι παραδείγματα προς μίμηση.

Ευχαριστώ πολύ για την συμπαράσταση και κατανόηση την Ευανθία, η οποία με βοήθησε στην επεξήγηση περιβαλλοντικών εννοιών και στην αποσφαλμάτωση του πηγαίου κώδικα που χρησιμοποιήθηκε.

Θα ήθελα να εκφράσω τις ευχαριστίες μου και στον κ. Δημήτριο Σκούτα, τον επιβλέποντα κατά την διάρκεια της συγγραφής της πτυχιακής αυτής εργασίας, για την εμπιστοσύνη που μου έδειξε, την πολύτιμη βοήθεια που παρείχε, την άριστη συνεργασία και τις στοχευμένες συμβουλές για την αντιμετώπιση όλων των θεμάτων που παρουσιάστηκαν. Με την αμέριστη συμπαράσταση του η εργασία αυτή ολοκληρώθηκε με επιτυχία.

Εύχομαι η πτυχιακή αυτή εργασία και οι ιδέες και λύσεις που περιγράφει, να αποτελέσουν βάση για περαιτέρω ανάπτυξη και βελτίωση αισθητήρων και δικτύων, με απώτερο σκοπό την καταγραφή, μελέτη και βελτίωση της ποιότητας ζωής σε αστικό περιβάλλον.

Acknowledgements

Let me express my sincere gratitude to the people that supported, inspired, and helped me before, during and after my studies.

This page intentionally left blank

Περίληψη

Η εκπόνηση της παρούσας διπλωματικής εργασίας έγινε υπό την επίβλεψη και την καθοδήγηση του Επίκουρου Καθηγητή κ. Δημητρίου Σκούτα, στα πλαίσια απόκτησης του προπτυχιακού διπλώματος Μηχανικών Πληροφοριακών και Επικοινωνιακών Συστημάτων του Πανεπιστημίου Αιγαίου κατά το ακαδημαϊκό έτος 2021-2022.

Στόχος της εργασίας είναι η μελέτη και υλοποίηση ενός ολοκληρωμένου συστήματος παρακολούθησης της ποιότητας του ατμοσφαιρικού αέρα και της ηχορύπανσης σε αστικό περιβάλλον με χρήση του μικροελεγκτή ESP32 και της τεχνολογίας LoRa και LoRaWAN.

Η συσκευή καταγραφής περιλαμβάνει τον μικροελεγκτή ESP32, έναν αισθητήρα ήχου για την καταγραφή της έντασης του ήχου, έναν πολύ-αισθητήρα θερμοκρασίας/υγρασίας/ατμοσφ. πίεσης/eCO2/TVOC για την καταγραφή της ποιότητας του ατμοσφαιρικού αέρα και άλλων παραμέτρων του, έναν αισθητήρα μέτρησης αιωρούμενων σωματιδίων PM2.5/PM10/PM1, μία οθόνη OLED και ένα LoRa RF chip για την μετάδοση των δεδομένων. Η καταγραφή και αποστολή των τιμών γίνεται ανά τακτά χρονικά διαστήματα στην διάρκεια μιας ημέρας. Η συσκευή στέλνει τα δεδομένα σε ένα LoRaWAN gateway το οποίο με τη σειρά του στέλνει τα δεδομένα προς επεξεργασία στην εθελοντική πλατφόρμα ”The Things Network”. Ο τελικός προορισμός των μετρήσεων είναι η πλατφόρμα ThingSpeak όπου τα δεδομένα προβάλλονται δημόσια και επεξεργάζονται με χρήση του εργαλείου MATLAB.

Το ερευνητικό έργο αυτό υλοποιήθηκε κατόπιν μελέτης αντίστοιχων λύσεων καταγραφής των ιδίων παραμέτρων ώστε να παραχθεί ένα προϊόν όπου θα βοηθήσει στην μελέτη και βελτίωση της διαβίωσης σε αστικό περιβάλλον, αφού έχει παρατηρηθεί και συσχετισθεί η ηχορύπανση και η ποιότητα του ατμοσφαιρικού αέρα με χρόνια προβλήματα υγείας των κατοίκων του περιβάλλοντος αυτού.

Λέξεις Κλειδιά: ESP32, LoRa, LoRaWAN, αισθητήρας ήχου, The Things Network, ThingSpeak, αισθητήρας ποιότητας αέρα, eCO2, TVOC, MATLAB, PM10, PM2.5, PM1, αστικό περιβάλλον, ποιότητα ζωής, αγροτικό περιβάλλον

Abstract

The preparation of this thesis has been under the supervision of Assistant Professor Mr. Dimitrios Skoutas in fulfilment of the requirements for the degree of Engineer of Information and Communication Systems Engineering in the University of the Aegean, of the Department of Information and Communication Systems Engineering (ICSD).

Aim of this thesis is to study and implement a monitoring system of the air quality and sound pollution in urban environment with the use of ESP32 microcontroller and the technology of LoRa and LoRaWAN.

The sensor node essentially consists of the ESP32 microcontroller, a sound sensor (microphone) to measure the sound level, a multi-sensor of temperature/humidity/pressure/eCO2/TVOC for measuring the air quality and other environmental parameters, an OLED screen for readings output, a particulate matter (PM2.5, PM10, PM1) sensor and a LoRa RF transceiver for sensor readings transmission. Multiple times per day, the sensors gather the values, and the node sends them to the LoRaWAN gateway, which in turn sends them to the "The Thins Network" platform. The destination of the readings is the ThingSpeak integration where they are plotted in public charts and further analyzed with MATALAB.

This thesis has been written after researching other implementations and solutions for monitoring the parameters of air quality and sound pollution which have been documented to associate with chronic health issues of urban environment citizens.

Keywords: ESP32, LoRa, LoRaWAN, sound sensor, The Things Network, ThingSpeak, air quality sensor, eCO2, TVOC, MATLAB, PM10, PM2.5, PM1, urban environment, quality of life, rural environment

Πίνακας περιεχομένων

Abbreviations

LoRa	Long Range
LoRaWAN	Long Range Wide-Area Network
TTN	The Things Network
CSS	Chirp Spread Spectrum
RF	Radio Frequency
MAC	Medium Access Control
CPU	Central Processing Unit
PM2.5	Particulate matter smaller than 2.5μm
PM1	Particulate matter smaller than 1μm
PM10	Particulate matter smaller than 10μm
eCO2	Equivalent calculated carbon-dioxide
TVOC	Total Volatile Organic Compound
WHO	World Health Organization
SPI	Serial Peripheral Interface
I2C	Inter-Integrated Circuit
I2S	Inter-IC Sound
CSV	Comma Separated Values
JSON	JavaScript Object Notation
XML	eXtensible Markup Language
HAT	Hardware Attached on Top
OS	Operating System
SaaS	Software as a Service
ISM	Industrial, Scientific and Medical

This page intentionally left blank

1

Introduction

1.1 Scope, background, and objectives

In the last few decades, the development of urban territories and the ongoing growth of industrialization has caused air and noise pollution to become major issues in cities of developing and developed countries. Urban area is more susceptible compared to rural because of heavy industry and means of transport which makes the city residents more exposed to the pollutants. Exactly this exposure has been associated with increased mortality and hospital admissions due to cardiovascular and respiratory diseases [1].

Excessive noise and daily exposure to it can seriously harm our health and interferes with daily activities. It can cause sleep disturbance, psychophysiological and cardiovascular effects, reduce performance and cause changes in social behavior [2].

The situation described above has brought to the surface the need for monitoring these two factors of wellbeing with various technological solutions. By monitoring noise and air pollution we can design solutions to decrease the effects they have and enhance the life of citizens. This need is exactly what has driven the development of this project.

The aim of this thesis is to develop an end-to-end solution that will gather environmental data, especially air quality and sound level, send them using the latest technology in IoT (Internet of Things) to a platform and make them available for viewing and further analysis.

The following figure shows a brief representation of the whole project:

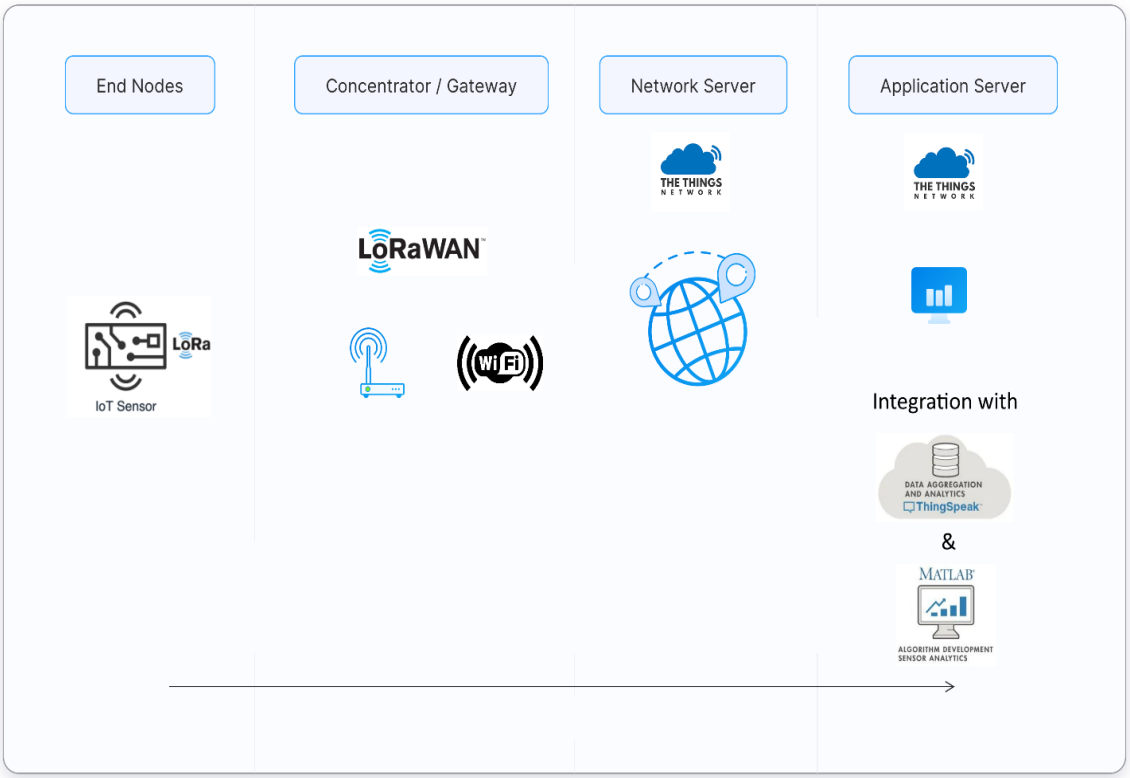


Figure 01 (source in Appendix I): Overview of thesis

1.2 Air quality

Air pollution represents a main threat to global society. The majority of people on the planet reside in locations where the World Health Organization's (WHO) recommended acceptable level for air quality is exceeded. Urban areas emit around 78 percent of all carbon emissions and significant airborne pollutants, which have a negative impact on more than 50 percent of the world's population that live there. Emissions, weather, and physicochemical changes are just a few of the many variables that can affect the regional air quality. Another significant factor that contributes to long-term air quality issues is urbanization, a process that modifies the size, makeup, and expansion of cities in response to the population surge.

In our thesis, we will measure the levels of eCO2 (Equivalent Calculated Carbon-Dioxide) in parallel with TVOC (Total Volatile Organic Compound) and PM (Particulate Matter).

Carbon dioxide (CO2) is a gas with no odor or color with present in the atmosphere on a regular basis. CO2 is a typical byproduct of biological metabolism and is present in exhaled breath. Additionally, it comes about as a result of burning fossil fuels and natural processes like volcanic eruptions. CO2 levels in outdoor air normally vary from 300 to 400 parts per million (ppm), but they can reach 600 to 900 ppm in urban areas. The way it affects the air quality is that whenever the

presence increases in the air, it also increases in the bloodstream, limiting the amount of oxygen available to the brain.

Generally, CO2 sensors are expensive. This drawback is solved by not measuring CO2 directly but by reporting eCO2. An eCO2 measurement is derived from a total volatile organic components (TVOC) measurement, which are compounds that have a high vapor pressure and low water solubility (Methane, alcohols, ketones, amines, and aromatic hydrocarbons etc.). The sensor we use has a metal oxide TVOC sensor integrated, and it works by measuring the resistance of a special reactive layer that is exposed to the air. The readings are used as inputs and after processing there is an estimation in CO2 concentration.

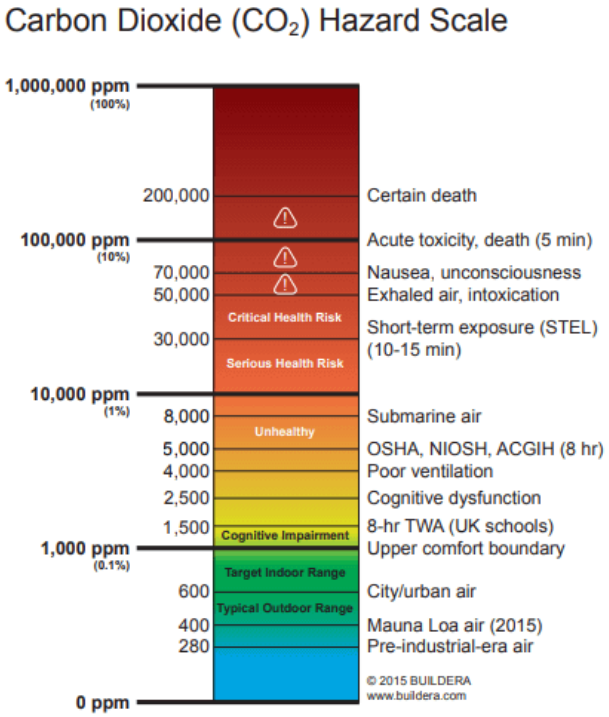


Figure 02 (source in Appendix I): CO2 hazard scale

TVOC Level [mg/m3]	TVOC Level [ppb]	Meaning
<0.25	56	Ideal
0.25-0.5	56-112	Good
0.5-1.5	112-336	Moderate
1.5-3.0	336-672	Elevated
>3.0	>672	High

Figure 03 (source in Appendix I): TVOC air quality scale

There are many pollutants, as many studies have shown, that are the main cause of human disease. Among them, particulate matter (PM), which are particles of variable but microscopic diameter that act as follows: they enter the respiratory system through inhalation and cause cardiovascular and respiratory diseases, central nervous system dysfunctions, reproductive dysfunctions, and cancer. The average annual concentration of fine particles with a diameter of less than 2.5 μm (PM2.5), in the most polluted cities is almost 20 times higher than in the cleanest city, surveys in 499 global cities have shown.

Particle size	Penetration degree in human respiratory system
>11 μm	Passage into nostrils and upper respiratory tract
7–11 μm	Passage into nasal cavity
4.7–7 μm	Passage into larynx
3.3–4.7 μm	Passage into trachea-bronchial area
2.1–3.3 μm	Secondary bronchial area passage
1.1–2.1 μm	Terminal bronchial area passage
0.65–1.1 μm	Bronchioles penetrability
0.43–0.65 μm	Alveolar penetrability

Type		PM diameter [μm]
Particulate contaminants	Smog	0.01–1
	Soot	0.01–0.8
	Tobacco smoke	0.01–1
	Fly ash	1–100
	Cement Dust	8–100
Biological Contaminants	Bacteria and bacterial spores	0.7–10
	Viruses	0.01–1
	Fungi and molds	2–12
	Allergens (dogs, cats, pollen, household dust)	0.1–100
Types of Dust	Atmospheric dust	0.01–1
	Heavy dust	100–1000
	Settling dust	1–100
Gases	Different gaseous contaminants	0.0001–0.01

Figure 04 (source in Appendix I): Particulate matter info and examples

The following figures show the limits and hazard scale for the PM concentration:

Qualitative name	Index	$PM_{2.5}$ (1-hr) μg/m³	PM_{10} (1-hr) μg/m³
Very high	>100	>110	>240
High	75-100	55-110	90-180
Medium	50-75	30-55	50-90
Low	25-50	15-30	25-50
Very low	0-25	0-15	0-25

Figure 05 (source in Appendix I): Common Air Quality Index (CAQI) and PM levels

Especially in Europe, the emission standards and limits are the following:

European Union [edit]

The European Union has established the European emission standards, which include limits for particulates in the air:[133]

European Air Quality Index	Good	Fair	Moderate	Poor	Very poor	Extremely poor
Particles less than 2.5µm ($PM_{2.5}$)	0-10 µg/m³	10-20 µg/m³	20-25 µg/m³	25-50 µg/m³	50-75 µg/m³	75-800 µg/m³
Particles less than 10µm (PM_{10})	0-20 µg/m³	20-40 µg/m³	40-50 µg/m³	50-100 µg/m³	100-150 µg/m³	150-1200 µg/m³

Figure 06 (source in Appendix I): European standards about emissions

1.3 Sound pollution

The spread of noise, commonly referred to as environmental noise or sound pollution, has a number of consequences on human or animal behavior, the most of which are at least somewhat harmful. The main sources are machinery, transportation, and propagation systems. Some of the main causes of noise in residential districts are loud music, transportation (traffic, rail, airplanes, etc.), construction, electrical generators, and people, which are amplified by poor urban design. According to the WHO, the typical permissible noise level in residential areas is 50 dB, which is far lower than the current average noise level of 98 dB in cities. High levels of noise are linked to cardiovascular effects, including an increased risk of coronary artery disease [3].

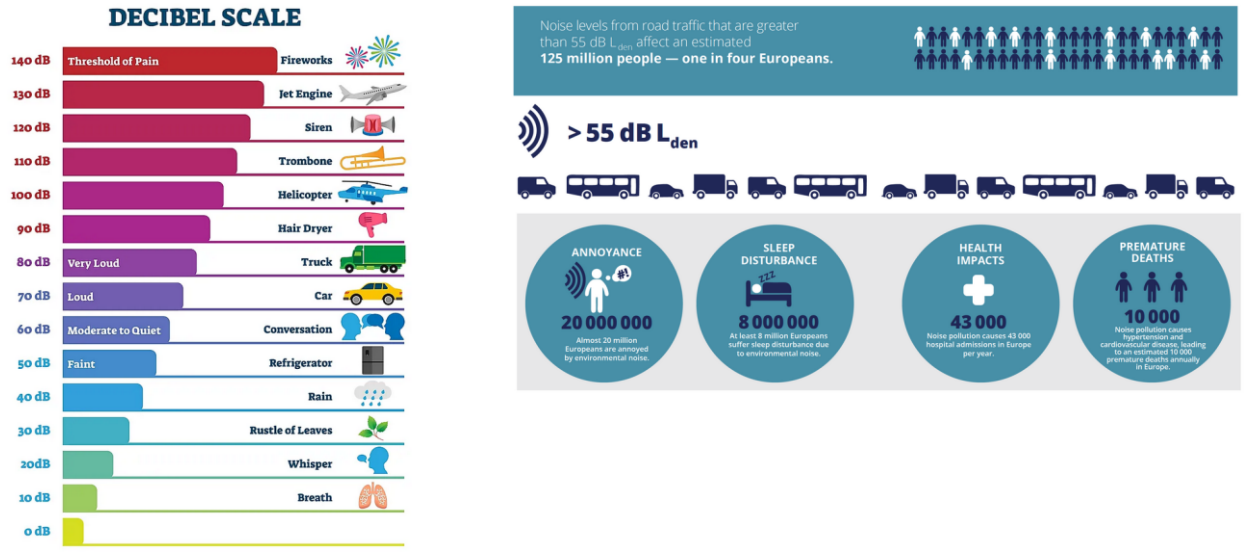


Figure 07, 08 (source in Appendix I): Sound levels and everyday examples

1.4 Motivation – Health issues and air quality/noise pollution

The motivation behind the whole project is simple and yet very important: there are some characteristics of the air we breathe and the sound levels we are exposed to that have direct impact to our health. This applies to every area we reside but especially in a city. Monitoring these indicators helps us decide for better measures and actions to improve our daily life and avoid exposure to threatening conditions. Such procedures are already in place in major cities around the world like China, where citizens are informed about the air quality and sound levels, and they are advised to wear a mask or avoid places with poor or hazardous conditions. So, our monitoring implementation can be a starting point and a basis for further scaling and enrichment in a city that wants to be smarter, proactive, and provide healthier environment to its citizens.

1.5 Thesis structure

Chapter 1 is the introduction of the thesis, in which we explain the motivation which is the health issues associated with air quality and noise pollution in the city, the contribution which is a complete monitoring solution of the environmental parameters we are interested in and the further description of them.

Chapter 2 is about the technologies we use to send the sensor readings from the end node to our gateway and to the platforms we will use for further analysis.

Chapter 3 is focused on describing all the parts and sensors of our LoRa end node which is responsible for acquiring the data and sending them using LoRa to our gateway.

Chapter 4 is focused on describing all the parts of our LoRaWAN gateway which is responsible for receiving the sensor readings from the end node and forwarding them to “The Things Network” platform.

Chapter 5 is describing the destination of the data we receive and include the “The Things Network” platform on which we create an integration with the “ThingSpeak” platform, and we publish their graphs.

Chapter 6 is about the analysis of the sensor data. There, with the help of MATLAB analysis we calculate important graphs and statistics.

Chapter 7 is about the evaluation of our monitoring solution, mentioning any improvements and any conclusions about the data we received.

2 Data transmission – LoRa, LoRaWAN

2.1 Description

For our thesis, we are making use of RF transmission with LoRa and LoRaWAN technology, which includes one end-node and one gateway.

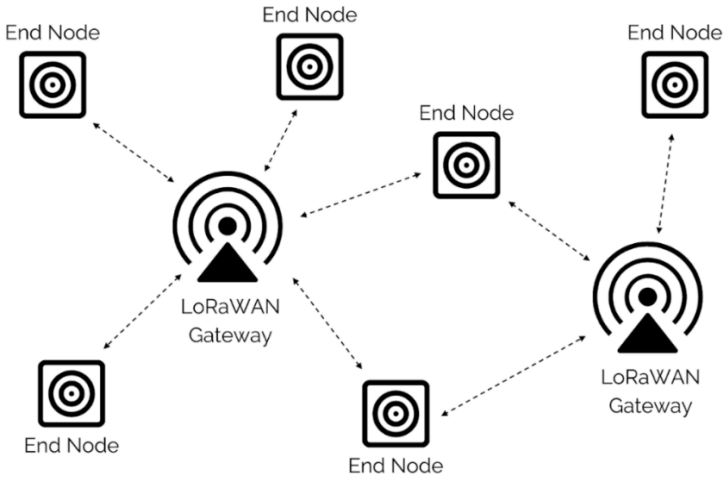


Figure 09 (source in Appendix I): Example of typical communication between end nodes and gateways

LoRaWAN gateways act as the link between end nodes and the network. To receive information from the End Nodes, gateways in the area are equipped with a very capable LoRa concentrator and are considered as a powerful router. The primary types of LoRaWAN Gateways are two and they distinguished based on their type of software running:

- Minimal firmware: runs only the packet forwarding software.

- Operating system: the software responsible for packet forwarding runs in the background. This gives the possibility of utilizing the gateway for other purposes and running more tools to provide flexibility and added value on the expense of power consumption.

End Nodes are devices that may be placed anywhere at the end of a network and are outfitted with sensors or actuators to gather and monitor data. Low-power microcontrollers are the most common version, as they may be deployed for years, are maintenance-free, have a small battery, and are equipped with a LoRa transmitter to deliver data packets to Gateways.

It is important to clearly understand that there is a difference in the type of communication between the entities of a LoRaWAN network: an end node communicates with near-by gateways through low power LoRaWAN, whereas gateways communicate with the Network Server using higher bandwidth communication protocols (Ethernet, WiFi, Cellular) [4].

All the parts of the ecosystem of our projects are explained in the following sections.

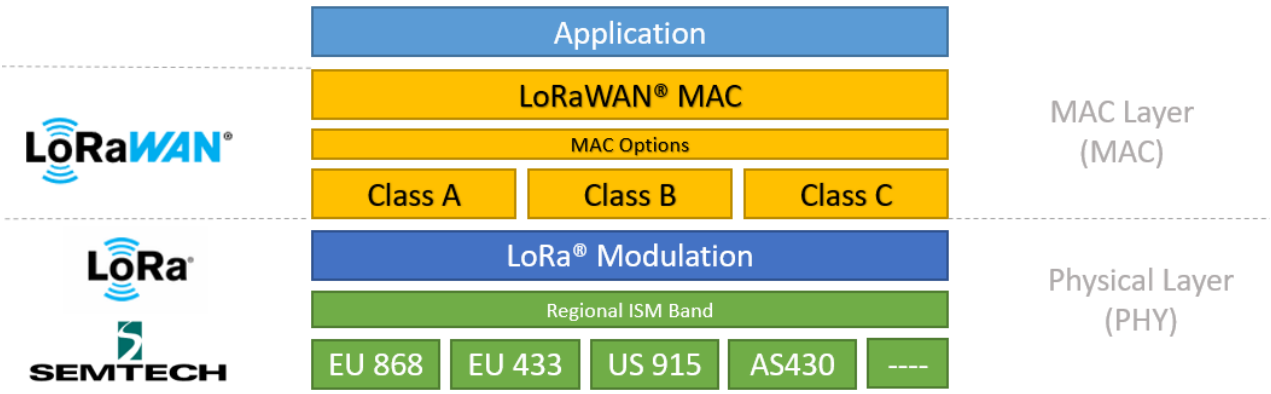


Figure 10 (source in Appendix I): Brief description of LoRa and LoRaWAN layer difference

2.2 LoRa

Figure 11 (source in Appendix I): LoRa logo

Long Range (LoRa) is a great wireless modulation technique that evolved from Chirp Spread Spectrum (CSS) technology, in which data is encoded in radio waves using chirp pulses. The breakthrough features for IoT applications are low power consumption, secure data transmission, and increased data propagation distance. An additional and groundbreaking feature of modulated LoRa transmission is its property to remain impervious to interference without any loss of information. This method is an amazing fit for use cases/applications that send small pieces of data at low bit rates and over a longer distance compared to WiFi, Bluetooth, or ZigBee. These characteristics make LoRa an excellent and brilliant choice for low-power sensors and actuators.

LoRa can operate in unlicensed sub-gigahertz bands such as 915 MHz, 868 MHz, and 433 MHz, for example. These frequencies are part of the ISM bands, which are used for industrial, scientific, and medical purposes around the world. Ranges vary by region. The United States, for example, uses the 915 MHz band, while Asia uses the 865 to 867 MHz and 920 to 923 MHz bands. It can also operate on the 2.4 GHz frequency for faster data rates than the sub-gigahertz band, but at the expense of range. [5].

Channel Plan	Common Name
EU863-870	EU868
US902-928	US915
CN779-787	CN779
EU433	EU433
AU915-928	AU915
CN470-510	CN470
AS923	AS923
KR920-923	KR920
IN865-867	IN865
RU864-870	RU864

Figure 12 (source in Appendix I): Common frequency plans

2.3 LoRaWAN

Figure 13 (source in Appendix I): LoRaWAN logo

LoRaWAN is a network protocol (media access control (MAC) layer) based on the innovative LoRa modulation technique and provides long-range, low-power and security features that are ideal for use cases involving, for example, telemetry and sensor data very small size as well as low power actuators. It is a software layer that defines how LoRa hardware transmits and receives, and the format of the messages sent. The LoRa Alliance is responsible for the development and maintenance of the LoRaWAN protocol.

In a typical LoRaWAN network, gateways carry messages between end nodes and a central network server in the backend, forming a star architecture. End devices communicate with one or more gateways (depending on which one is in range and with the strongest signal) using LoRa, while the gateways communicate with the network server via regular IP connections. Although uplink messages from an end device to the network server are supposed to be the primary traffic, all communication is usually bidirectional, sometimes with the presence of downlink messages.

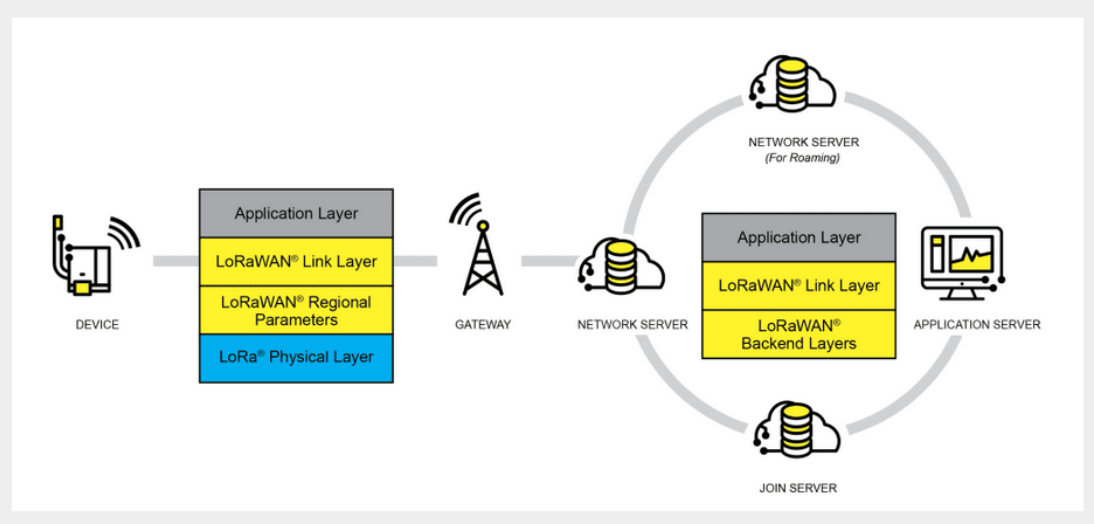


Figure 14 (source in Appendix I): LoRaWAN Network Architecture

A typical LoRaWAN network, as we can see in the previous figure, includes the following entities:

- End devices: these are sensors or actuators, of any size but low power, as they use batteries (unless connected to a continuous power supply) that send LoRa-modulated wireless messages to gateways (uplink message) or receive messages back (downlink message).
- Gateways: receive messages from end devices and forward them to the network server. Also responsible for sending messages back to the end nodes. They are usually connected to a continuous power supply.
- Network server: software running on a private server (or cloud platform) that is responsible for managing the entire network.
- Application servers: a piece of software running on a private server (or cloud platform), responsible for securely processing application data and for viewing or sending it to other applications for further processing.
- Join Server: a piece of software running on a private server (or cloud platform) that processes end device join request messages so that their messages can successfully reach the network and application servers and applications.

The end nodes are split into 3 categories based on energy consumption and message transmission [6]:

- Class A (Bi-directional): This class supports bidirectional communication, with each end-device's uplink broadcast followed by two short downlink reception windows. It is the most energy-efficient system for applications that only require downlink communication with the server once an uplink transmission has been sent. At any other time, downlink communications from the server will have to wait until the next planned uplink.
- Class B (Bi-directional with scheduled receive slots): This class allows for more receive slots. Class B devices, in addition to the random receive windows of Class A, open additional receive windows at predetermined periods.
- Class C (Bi-directional with maximal receive slots): This class has practically always open receive windows, which are only closed when sending. Class C end-devices consume more power than Class A or Class B (thus they are usually connected to a continuous power supply), but they have the shortest server-to-end-device communication latency.

Each gateway is connected using IP connections to the network server, and end devices communicate with gateways within range rather than a specific one, since the LoRaWAN networks use the ALOHA protocol. The Network Server receives the messages and then message deduplication is used: if the Network Server receives several copies of the same message, it maintains just one copy and discards the others [7]

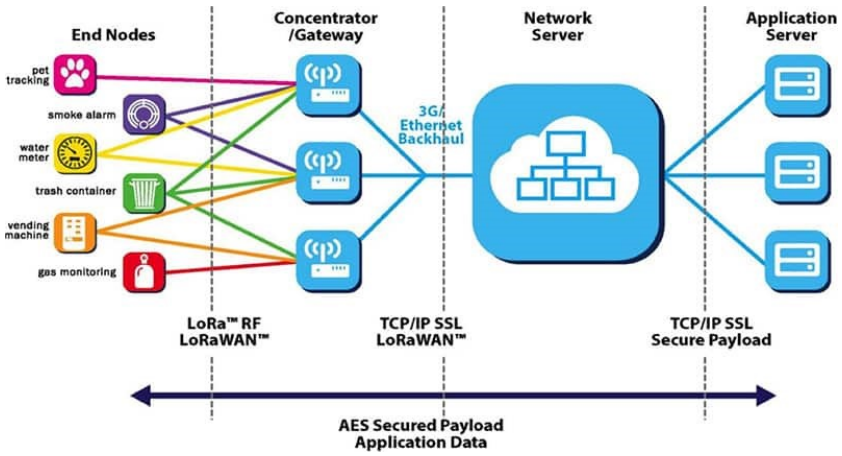


Figure 15 (source in Appendix I): Typical LoRaWAN network

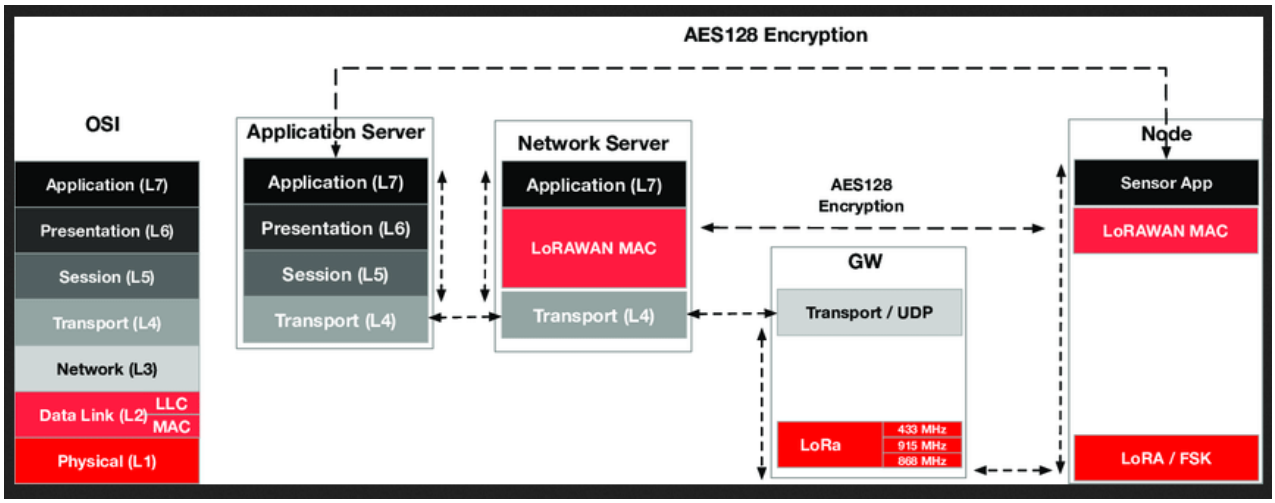


Figure 16 (source in Appendix I): LoRaWAN OSI reference

The following are some key characteristics and benefits of using this technology:

- Ultra-low power: Low power optimized end devices that can last large periods of time on small batteries.
- Long range: In rural locations, gateways may broadcast and receive signals over up to 10-11 kilometers, and in urban areas, up to 3-4 kilometers.
- Deep indoor penetration: Multi floor buildings can easily get coverage.
- License free spectrum: The technology operates in the license-free frequency spectrum.
- Geolocation: By using triangulation and the position of at least 3 gateways, the location of any kind of end device can be determined without the need for Global Positioning System (GPS)
- High capacity: Vast amounts of messages from thousands of gateways can be handled by Network Servers.
- Public and private deployments: Public and private LoRaWAN networks can be easily deployed using the same type of hardware (gateways, actuators, end devices, antennas) and pieces of software (packet forwarders, LoRaWAN software stacks).
- End-to-end security: Highly secured communication (AES-128 encryption support) is ensured between the end device and the application server.
- Over the air (OTA) firmware updates: Remote update of the firmware can be achieved (microcontroller code and the LoRaWAN stack) for one or group of end devices with no need of on-premises access.
- Roaming: End devices can use any gateway within range and perform handovers from one network to another without any issue.
- Low cost: Minimal-cost end nodes and sensors, minimal infrastructure, and open-source software usage.
- Certification program: LoRa Alliance certifies end devices and guarantees about the reliability of the devices and LoRaWAN specification compliance.

- Ecosystem/Community: LoRaWAN features a wide ecosystem of actuator/device/gateway/antenna manufacturers, network service providers, and application developers, as well as excellent support in the event of a problem.

The following are a small number of use cases with ways of application to give some insight [8]:

- Vaccine transport monitoring: Temperature and humidity sensors are employed, to guarantee ideal conditions for the vaccines while in transit.
- Animal conservation: Managing endangered species using tracking sensors.
- Elderly patients monitoring: Wristband sensors provide fall detection and panic buttons provide emergency awareness.
- Smart Metering: Providing efficient solutions to monitor energy consumption at an asset level.
- Smart farms: Crop soil moisture is continuously monitored, and watering schedules are optimized.
- Food safety: Real time temperature monitoring for food quality maintenance.
- Smart Grid: Real-time monitoring of electricity networks and immediate detection, reaction, and pro-action to changes in usage and any issues.
- Smart city: Waste bin level alerts, air quality, sound pollution, damage report can be automated and sent to the responsible staff/committee to prioritize/optimize solutions.
- Efficient workspaces: Monitoring of room occupancy, temperature, humidity, energy use, and parking availability for optimal scheduling.
- Home, Building, Industrial automation: Excellent use for low power automation with incredible scaling.
- LoRa in space: Full scale Earth LoRaWAN-based coverage with satellites.

3

Measurements acquisition – LoRa end node

3.1 Description

The heart of our project is a LoRa end node, and the next picture shows the hardware that was used and combined to build it:

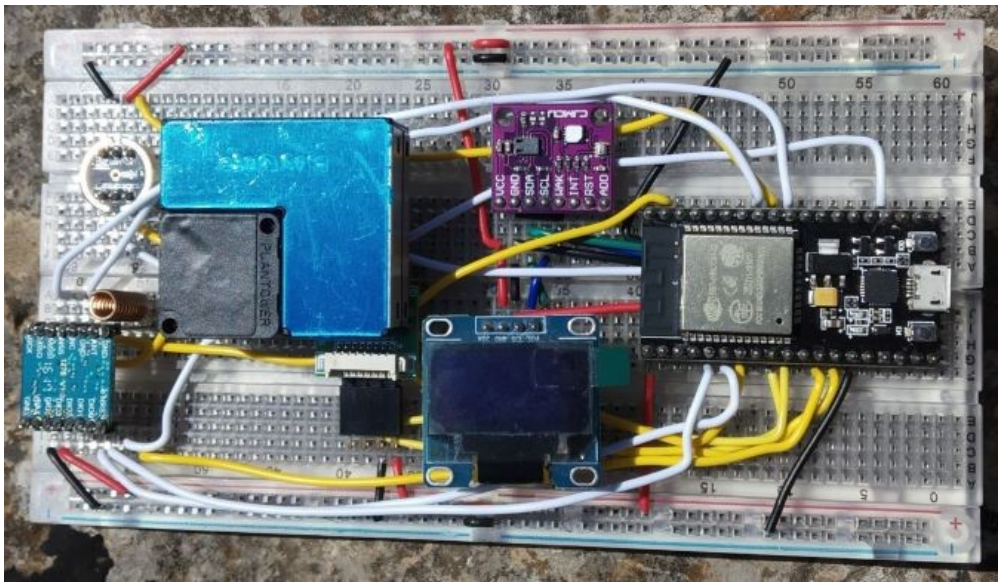


Figure 17 (source in Appendix I): End node with sensors

This node is where everything begins. It is responsible for acquiring the values from the sensors and transmitting them to the cloud platforms that we use for representation and calculations. It is battery powered with low power consumption.

3.2 Hardware involved

In the next subsections, we will enumerate and describe all the components that make up the LoRa node that was developed and used for our project:

3.2.1 ESP32

Our main MCU (MicroController Unit) is ESP-32 (by Espressif). It is an MCU that combines WiFi and Bluetooth technologies along with a powerful CPU (Central Processing Unit) and more than enough RAM (Random Access Memory) and ROM (Read Only Memory) [9].

- 11b/g/n (802.11n, speed up to 150Mbps)
- WIFI Frequency Range 2.4GHz ~ 2.5GHz
- Clock frequency adjustment range from 80 MHz to 240 MHz, support for RTOS
- Built-in 2-channel 12-bit high-precision ADC with up to 18 channels
- Support UART/GPIO/ADC/DAC/SDIO/SD card/PWM/I2C/I2S interface
- Support multiple sleep modes, ESP32 chip sleep current is less than 5 μA
- Embedded Lwip protocol stack
- Supports STA/AP/STA + AP operation mode
- Supports remote firmware upgrade (FOTA)
- General AT commands can be used quickly
- Support secondary development, integrated Windows, Linux development environment

Figure 18 (source in Appendix I): ESP32 microcontroller

This MCU was chosen because of the capabilities it offers and the possibility for further expansion of the project. It is programmed in the Arduino programming language, which is based in Processing programming language, a very simple hardware programming language, which is like C/C++ language. The ESP32 has a very good amount of memory for programming which we need since we use many libraries for our sensors to work and to send data using “The things Network” libraries.

```
Sketch uses 323758 bytes (24%) of program storage space. Maximum is 1310720 bytes.
Global variables use 16796 bytes (5%) of dynamic memory, leaving 310884 bytes for local variables. Maximum is 327680 bytes.
```

Figure 19 (source in Appendix I): Arduino IDE message on memory allocation, after compilation of code

3.2.2 SSD1306 - OLED

SSD1306 is a CMOS OLED screen. The resolution of the screen we used is 128x64 and uses the I2C interface. The module is integrated with contrast control, oscillator, and RAM, which makes the number of external components and power consumption minimum [10].

Figure 18 (source in Appendix I): OLED screen

In this project, it is used for debugging and as a mean to show the sensors' readings and their transmission. In the code we used and adjusted the example from the included library (https://github.com/greiman/SSD1306Ascii).

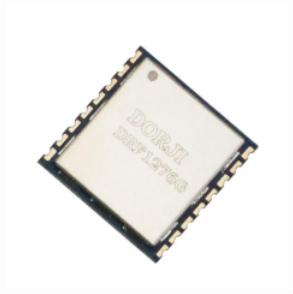
3.2.3 DRF1276G – LoRa RF chip

DRF1276G is a LoRa module that consists of SX1276 chip, SMD crystal and antenna. It operates at low voltage (less than 3.6V) with standby current that reaches the levels of μA which makes it suitable for battery-operated use cases/applications [11].

DRF1276G
20dBm LoRa Long Range RF Front-end Module **V1.20**

Features:

- Frequency Range: 868/915MHz
- Modulation: FSK/GFSK/MSK/LoRa
- SPI Data Interface
- Sensitivity: -139dBm
- Output Power: +20dBm
- Data Rate: <300 kbps
- 127dB dynamic Range RSSI
- Excellent blocking immunity
- Preamble detection
- Automatic RF sense and CAD monitor
- Built-in bit synchronizer for clock recovery
- Packet engine up to 256 bytes with CRC
- Working Temperature: -40°C ~+80°C
- Build-in temperature sensor
- Standby current: ≤ 1uA
- Supply voltage: 1.8~3.6V

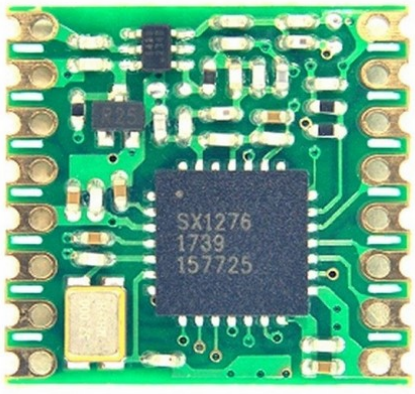


Applications

- Remote Control
- Smart metering
- Home Automation
- Personal data logger
- Wireless sensor network
- Remote keyless entry
- Wireless PC peripherals

Figure 21 (source in Appendix I): LoRa module

The SX1276 transceiver included in the module feature the LoRa modem which provides ultra-long range spread spectrum communication and strong interference immunity while consuming the least amount of energy. Using Semtech's unique, patented and innovative LoRa modulation technology, SX1276 can achieve a very high sensitivity (greater than -148dBm) [12] [13].



Features

- LoRa Modem
- 168dB maximum link budget
- +20dBm - 100 mW constant RF output vs. V supply
- +14dBm high efficiency PA
- Programmable bit rate up to 300kbps
- High sensitivity: down to -148dBm
- Bullet-proof front end: IIP3 = -11dBm
- Excellent blocking immunity
- Low RX current of 9.9mA, 200nA register retention
- Fully integrated synthesizer with a resolution of 61Hz
- FSK, GFSK, MSK, GMSK, LoRa and OOK modulation
- Built-in bit synchronizer for clock recovery
- Preamble detection
- 127dB Dynamic Range RSSI
- Automatic RF Sense and CAD with ultra-fast AFC
- Packet engine up to 256 bytes with CRC
- Built-in temperature sensor and low battery indicator

Applications

- Automated Meter Reading
- Home and Building Automation
- Wireless Alarm and Security Systems
- Industrial Monitoring and Control
- Long range Irrigation Systems

Figure 22 (source in Appendix I): LoRa modem

With this module we achieve data transmission from our end node to our gateway. In the code we used and adjusted the example from the included library (https://github.com/manuelbl/ttn-esp32).

3.2.4 CJMCU8128 – Air quality board

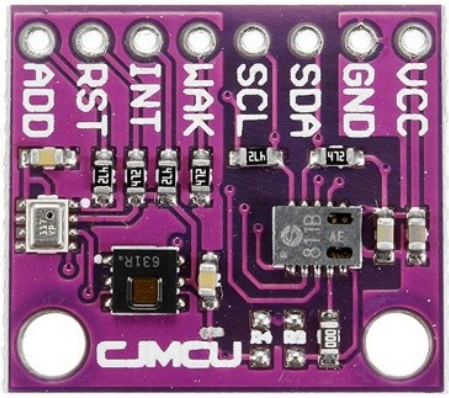


Figure 23 (source in Appendix I): TVOC, eCO2, temperature, humidity sensor

This board consists of 3 chips:

CCS811 (by ScioSense): It is a very energy efficient digital gas sensor that uses the I²C interface. The use of a metal oxide (MOX) gas sensor ensures the easier and more accurate detection of a wide range of Volatile Organic Compounds (VOCs) for outdoor/indoor air quality monitoring. By using this incredibly thin hotplate technology, a highly reliable solution for gas sensors, very fast cycle times and a significant reduction in energy consumption is achieved [14]

This sensor will measure eCO2 (equivalent calculated Carbon-dioxide) concentration (400 to 8192 parts per million (ppm)), and TVOC (Total Volatile Organic Compound) concentration (0 to 1187 parts per billion (ppb)). The range of detections involves Alcohols, Ketones, Organic Acids, Amines, and various Hydrocarbons [15]. In the code we used and adjusted the example from the included library (https://github.com/adafruit/Adafruit_CCS811).

HDC1080 (by Texas Instruments): It is a highly accurate Digital Humidity sensor which integrates a temperature sensor operating at very low power and is already factory calibrated [16]. In the code we used and adjusted the example from the included library (https://github.com/adafruit/Adafruit_Si7021).

BMP280 (by Bosch): It is a barometric pressure sensor, which is the best solution for small size related applications. Its small size and its low power consumption allow for the implementation of devices where the use case demands the use of battery. The device is packaged and already factory-optimized in terms of power consumption, resolution, and filtering [17]. In the code we used and adjusted the example from the included library (https://github.com/adafruit/Adafruit_BMP280_Library).

We use this group of sensors as our main source of retrieving environmental variables data.

3.2.5 PMSA003 - particulate matter sensor

Figure 24 (source in Appendix I): PM1, PM2.5, PM10 sensor

This sensor is a universal particle concentration sensor, which measures the number of air-suspended particles, i.e., the air concentration of particles, and renders them in the form of digital interface. For such sensor, use of laser scattering principle is applied, which entails producing scattering by using laser to radiate air-suspended particles, collecting scattering light in a predefined degree, and ultimately obtaining the scattering light change over time curve. Finally, equivalent particle diameter and the arithmetic amount of particles with different diameter per unit volume can be calculated by the integrated microprocessor based on Mie scattering theory [18].

The values that we retrieve from this sensor are PM1, PM2.5, PM10 which refer to particles less than 1μm, 2.5 μm, 10 μm in diameter respectively. In the code we used and adjusted the example from the included library (https://github.com/plerup/espsoftwareserial, https://how2electronics.com/interfacing-pms5003-air-quality-sensor-arduino/).

3.2.6 INMP441 – Microphone

Figure 25 (source in Appendix I): Microphone used as sound level sensor

The INMP441 is a MEMS microphone with the following characteristics: high-performance, low power, digital-output and omnidirectional. It supports signal conditioning, integration of an analog-to-digital converter, implementation of anti-aliasing filters, supports power management, and an 24-bit I²S interface [19].

We used this microphone as a mean to measure the pressure level of sound with the appropriate use of the library (https://github.com/ikostoski/esp32-i2s-slm).

The end node was installed for 7 days in the heavily industrial city of Elefsina, Attica, Greece. This urban territory is surrounded by many factories, an oil refinery plant, and a national toll station. It is also a logistics center with many companies operating there, which leads to increased levels of traffic and therefore increased sound levels and air pollutants.

As a comparison, after the end of the data acquisition period in the city, we moved the end node to a rural area of the city of Mandra, away from factories and heavy traffic, again for 7 days. This relocation was necessary to have some readings from a cleaner area, healthier for the habitants.

The interval of the sensor reading, and data transmission is 1 per hour, so we have 24 readings per day. Then the readings are transmitted to the gateway and then to the network/application server.

4

LoRaWAN gateway

4.1 Description

Figure 26 (source in Appendix I): Complete gateway (Raspberry Pi 3 + RAK831)

Our LoRaWAN gateway is the receiver of the sensor readings from our end node and the responsible router of these messages to the network server of "The Things Network" platform.

Essentially it is a Raspberry Pi with a LoRaWAN concentrator hat (hardware attached on top) running the packet forwarder software to send the values to the Network Server.

4.2 Hardware involved

In the next sections, we will enumerate and describe all the components that make up the gateway that was developed and used for our project:

4.2.1 Raspberry Pi

Figure 27 (source in Appendix I): Raspberry Pi 3

The Raspberry Pi is a 4-core Linux (mainly) development board, with a GPIO (General Purpose Input/Output) header to connect actuators/sensors, and the ability to easily add any multimedia device (external microphone or camera) or any HAT (Hardware Attached on Top) to further develop projects of any use-case.

It is a typical and often to see piece of hardware in LoRaWAN gateways as it is robust, easily managed and very expandable. We are using an OS image provided by RAKWireless (manufacturer of the LoRaWAN HAT we use) which made our job to get the gateway up and running much easier. The OS is based on Raspberry Pi OS Debian Ver.10 “Buster” (at the time of writing) with all the required tools and configurations implemented.

4.2.2 RAK831

Figure 28 (source in Appendix I): RAK831 Raspberry Pi 3 HAT

RAK831 is a concentrator module, designed for a wide variety IoT use cases. It's a multi-channel high-performance transceiver module that can receive several LoRa packets at the same time using different spreading factors on multiple channels. As a complete RF front-end, it allows for reliable communication between the gateway and a great amount of LoRa end-nodes spread out across a large area. For proper operation, RAK831 needs a host system. It can receive up to 8 LoRa packets at the same time sent with varying spreading factors on various channels.

The module includes Semtech's SX1301 which is a digital baseband chip that incorporates a very capable digital signal processing engine that is specifically developed to give revolutionary gateway capabilities in the free ISM bands around the world. The LoRa concentrator IP is integrated into the SX1301 [20].

The software installation and whole setup of the gateway was easy thanks to the OS (Operating System) image provided by RAKWireless (https://downloads.rakwireless.com/LoRa/RAK2247-Mini-PCIe/RPi-Firmware/).

The gateway was temporarily placed at a height of 4m and within line of sight from the end node. The temporary placement was because we would move both the end node and the gateway after 1 week from the urban area to the rural area to get readings for further analysis and comparison between them.

5

Sensor data visual display

5.1 Description

For this implementation we decided not to implement a private LoRaWAN network but to use the collaborative platform of TTN (The Things Network) to gather all the sensor readings and then process them using some of the many integrations offered.

5.2 The Things Network

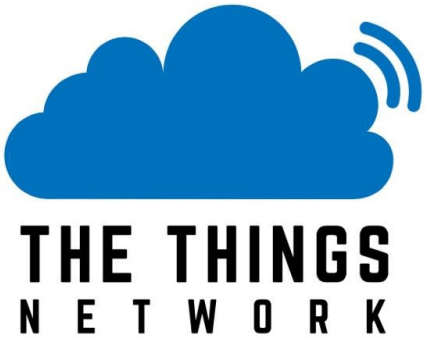


Figure 29 (source in Appendix I): TTN (The Things Network) logo

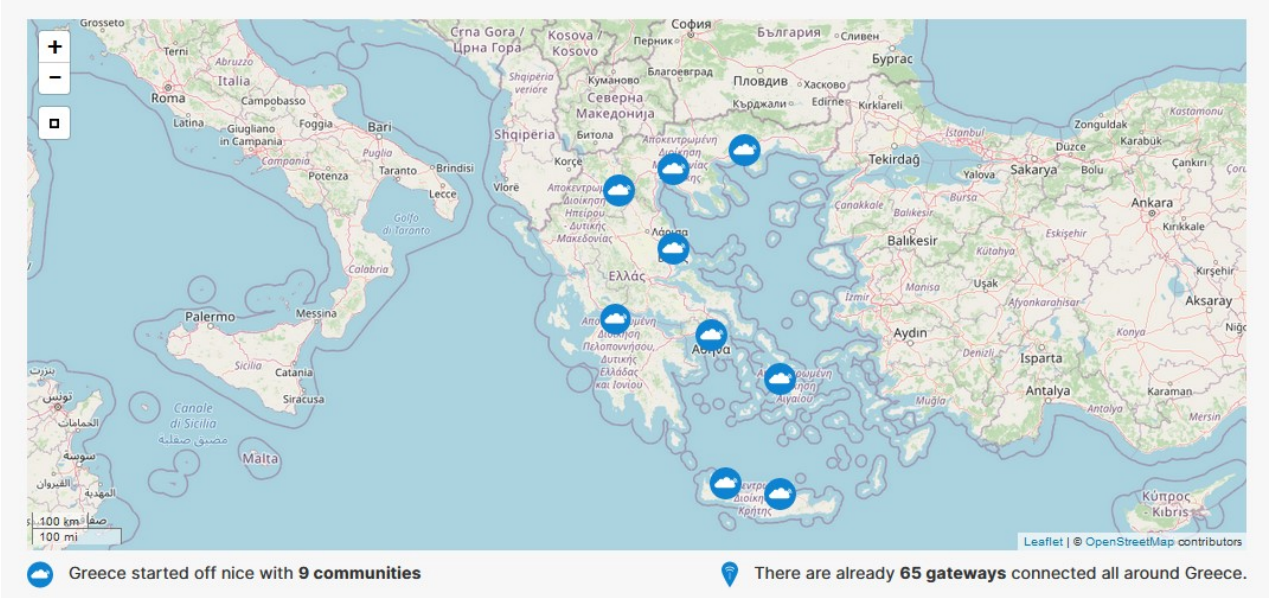


Image 30 (source in Appendix I): Greece TTN coverage overview

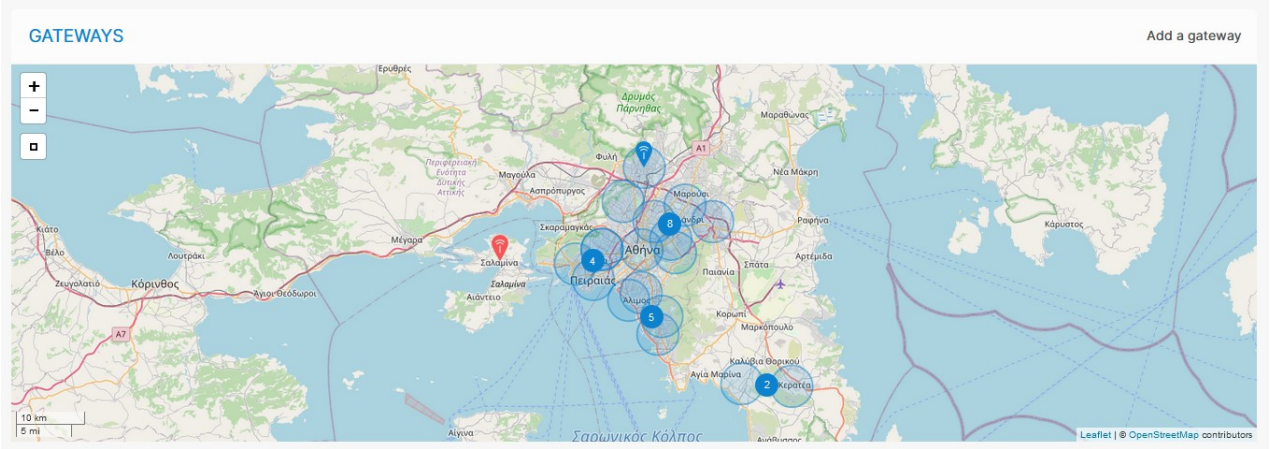


Figure 31 (source in Appendix I): Wider area of Attica, Greece, TTN gateway coverage

"The Things Network" SaaS (Software as a Service) platform is a global collaborative IoT ecosystem which creates networks and solutions using LoRaWAN technology. This initiative started in Denmark but quickly spread across the globe because of the support of all the people and organizations that wanted to use this new technology. It offers a set of open tools that can help someone build an IoT application at low cost but with security and the ability to scale.

5.3 App integration with ThingSpeak

Until now we have described how to get our data from the sensors to the The Things Network Network server. From there, by creating a Webhook integration, we will send the data to ThingSpeak platform for graph representation and further processing in MATLAB.

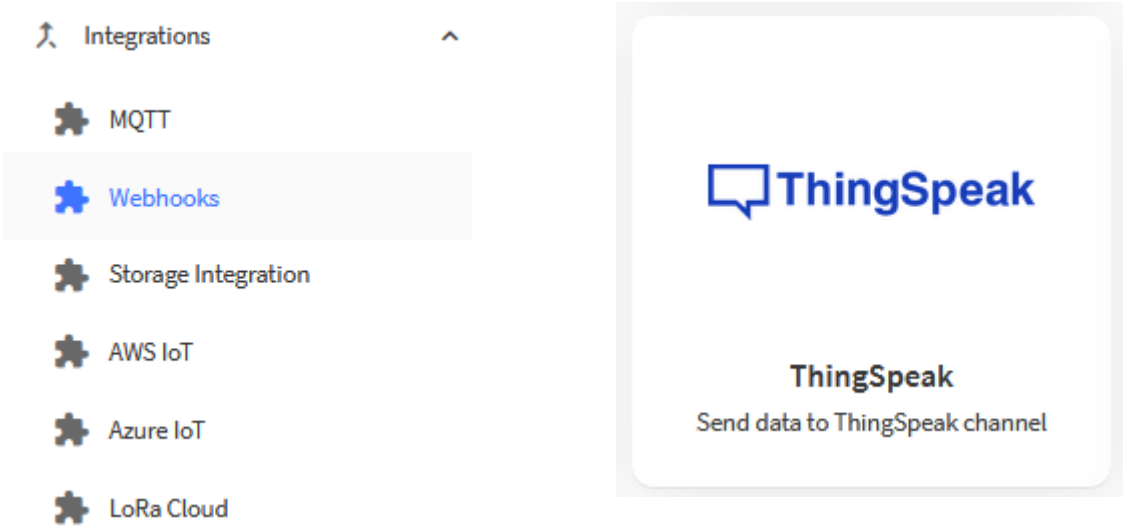


Figure 32 (source in Appendix I): ThingSpeak webhook in TTN console

5.4 ThingSpeak

Figure 33 (source in Appendix I): ThingSpeak logo

ThingSpeak is an IoT analytics platform (written in Ruby) that allows anyone to aggregate, view, and analyze real-time data streams in the cloud. Live data can be sent by any device, and instant visualization can be created, or alerts can be sent. One of the most important assets we are offered, and use is the integrated support for the numerical computing software MATLAB from MathWorks, which allows us to analyze and visualize uploaded data using MATLAB without having to purchase of a MATLAB license [1].

In the platform, we have created a channel that accepts the values from the LoRa end node and charts have been added for every field which is a sensor reading:

Figure 34 (source in Appendix I): ThingSpeak channel charts per field/sensor reading

As previously mentioned, the interval of the readings is once per hour. Every reading represents a dot in the graph. The graph shows all data for the last 30 days. They can also be exported as CSV, JSON and XML and imported to any other platform or software that we want for further analysis.

6

Sensor data analysis

6.1 Description

As mentioned earlier, the end node was placed for 5 days in 2 different areas, a rural and an urban. This gave us the opportunity to further analyze the data and make comparison between them.

6.2 MATLAB

Figure 35 (source in Appendix I): MATLAB logo

MATLAB is a numeric computing environment that besides other functionalities, it offers statistical analysis and plotting of functions and data. We use this functionality in our thesis for plotting the sensor data and computing certain values such as min, max, mean values, and more.

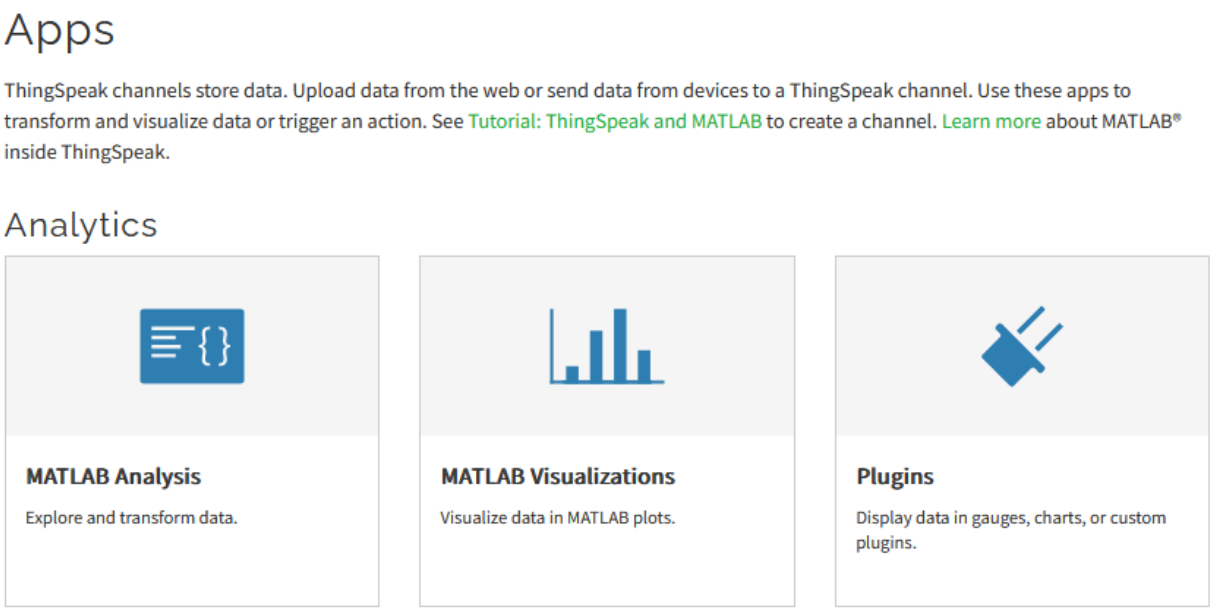


Figure 36 (source in Appendix I): MATLAB Apps in ThingSpeak

First, we added a visualization (code in Appendix IV) that shows all the readings in separate plots in addition to the ThingSpeak charts, for both areas:

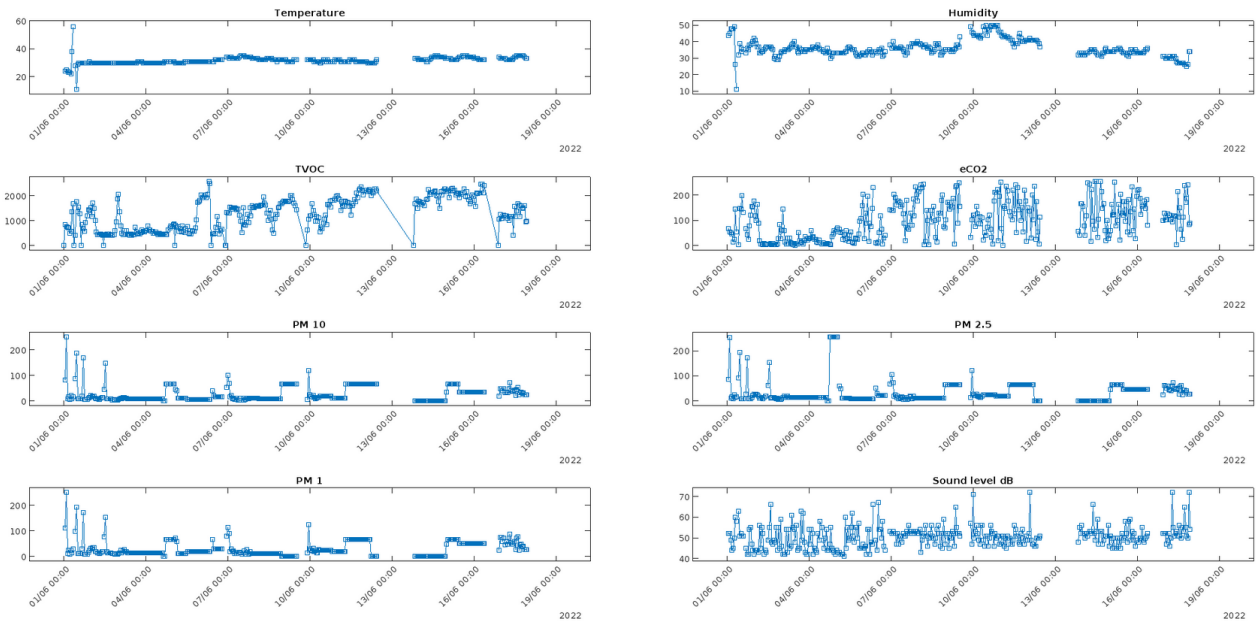


Figure 37 (source in Appendix IV): MATLAB Visualization in ThingSpeak

Then we developed some analysis code that calculates the min, max, median, mean values per day for every reading:

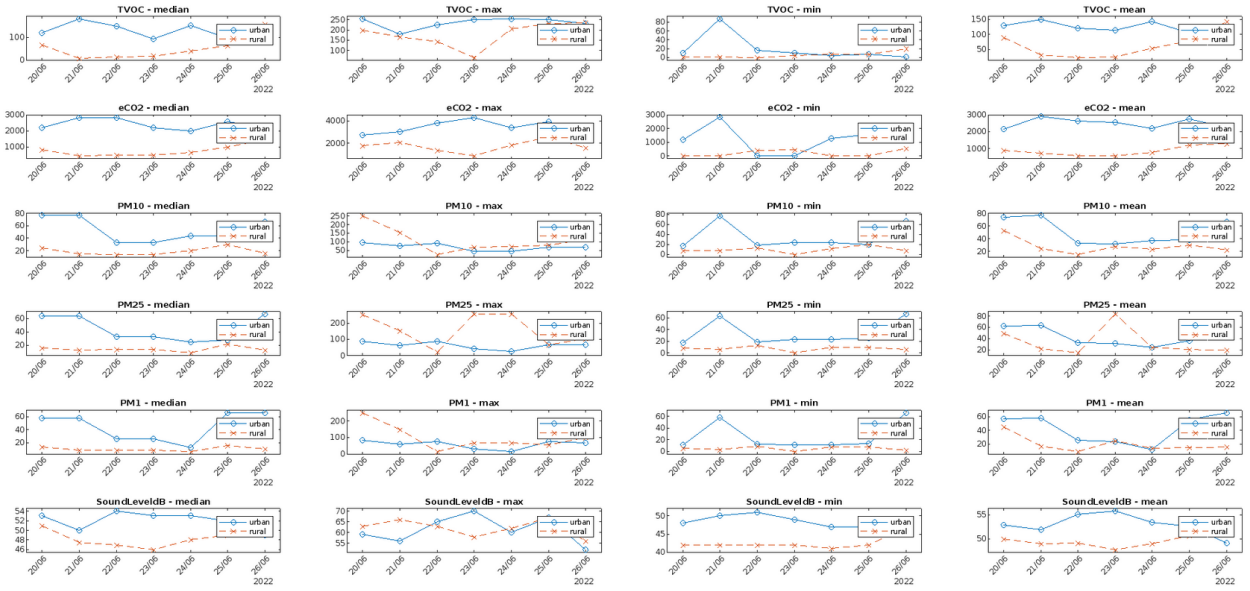


Figure 38 (source in Appendix I): MATLAB Visualization in ThingSpeak of all readings

Next, we created individual Visualizations for every sensor, and we calculated the same statistics plus a mean value of the 7-day period that we gathered data.

The following graphs and statistics are about the TVOC and eCO2 values:

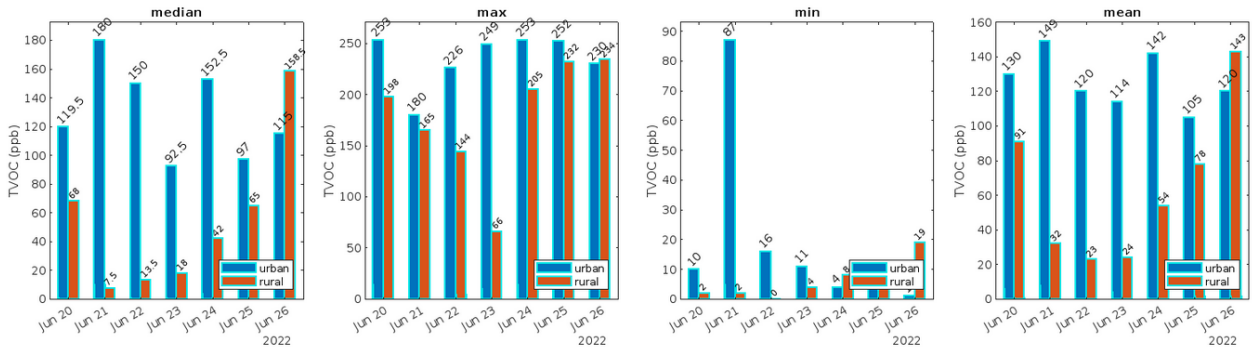


Figure 39 (source in Appendix I): MATLAB Visualization in ThingSpeak of TVOC readings

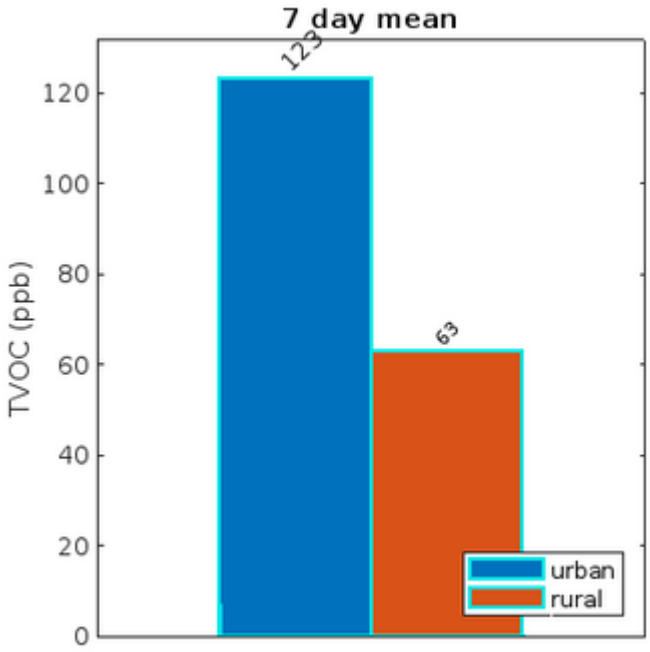


Figure 40 (source in Appendix I): MATLAB Visualization in ThingSpeak of TVOC 7-day period

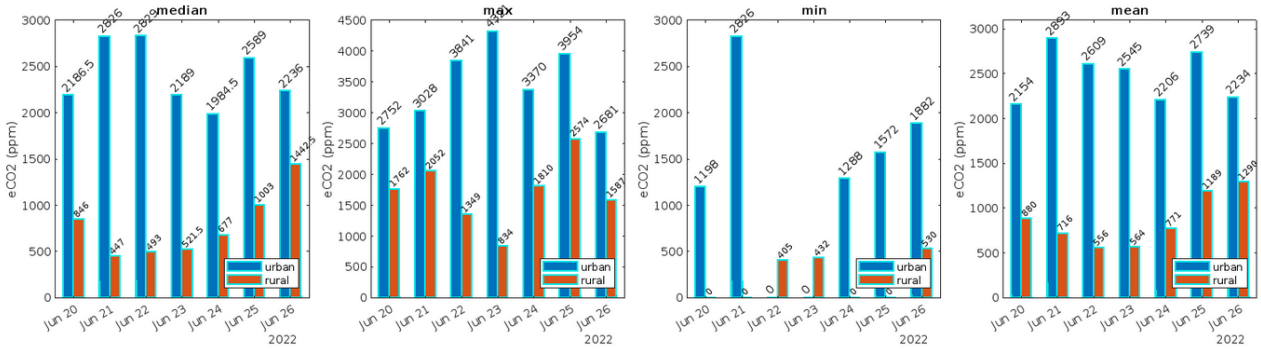


Figure 41 (source in Appendix I): MATLAB Visualization in ThingSpeak of eCO2 readings

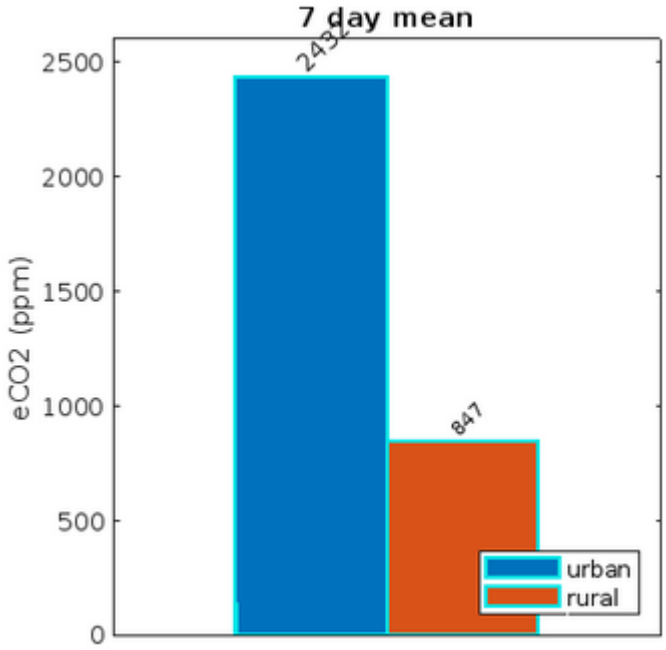


Figure 42 (source in Appendix I): MATLAB Visualization in ThingSpeak of eCO2 7-day period

As we can see from the values and graphs, during the 7-day period the mean value of TVOC for the urban area was **123 ppb** (parts per billion) which classifies the air quality as "Moderate" according to Chapter's 1.2 Figure. The rural area mean value was **63 ppb** which classifies it as "Good".

We can say with certainty that the results are as expected and a good indication that our sensors are working properly and sense the correct trend in air pollution.

The next graphs about eCO2 values during the 7-day period show a mean value of **2432 ppm** (parts per million) for the urban area which classifies the air quality as "Mediocre" according to Chapter's 1.2 Figure. The rural area mean value was **847 ppm** which classifies it as "Fair".

We can see that the urban values are elevated and there is a big difference between them and the rural ones. Again, the difference is expected, and the pollution trend was measured correctly.

The following graphs and statistics are about the PM10, PM2.5 and PM1 values:

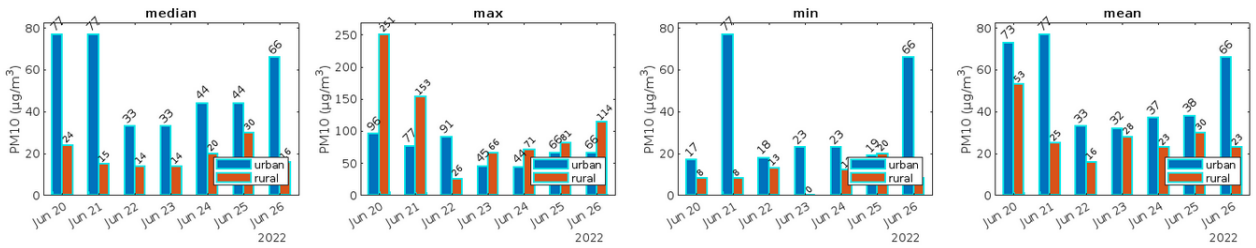


Figure 43 (source in Appendix I): MATLAB Visualization in ThingSpeak of PM10

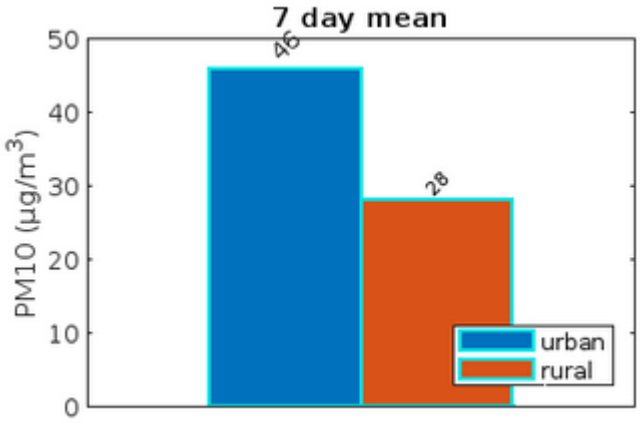


Figure 44 (source in Appendix I): MATLAB Visualization in ThingSpeak of PM10 7-day period

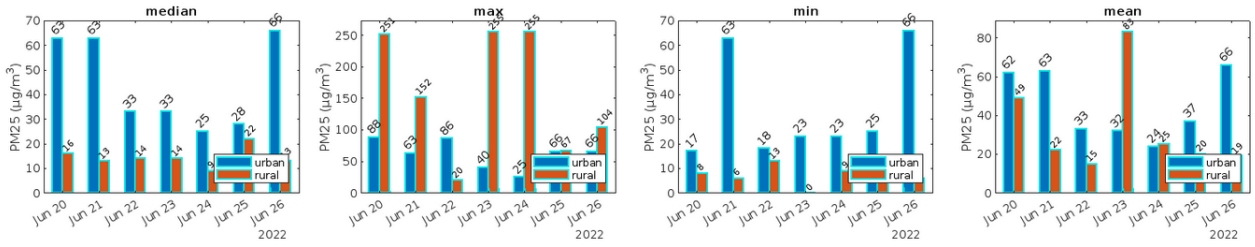


Figure 45 (source in Appendix I): MATLAB Visualization in ThingSpeak of PM2.5

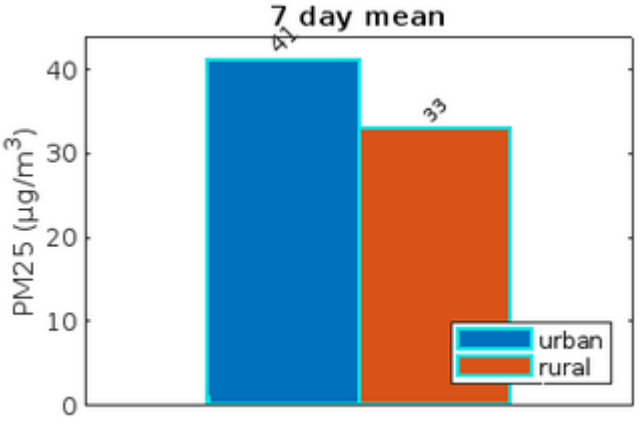


Figure 46 (source in Appendix I): MATLAB Visualization in ThingSpeak of PM2.5 7-day period

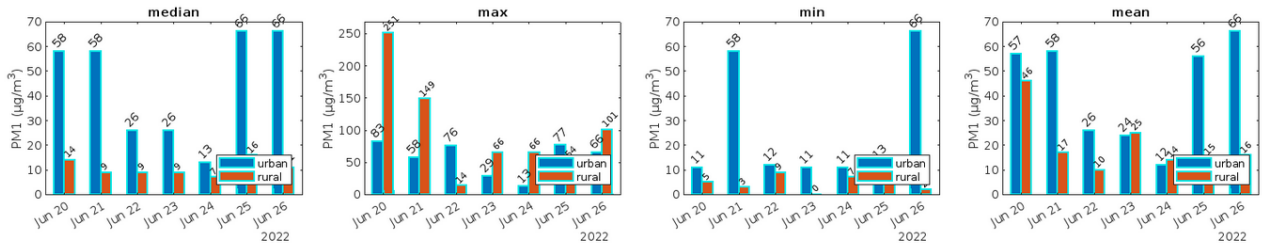


Figure 47 (source in Appendix I): MATLAB Visualization in ThingSpeak of PM1

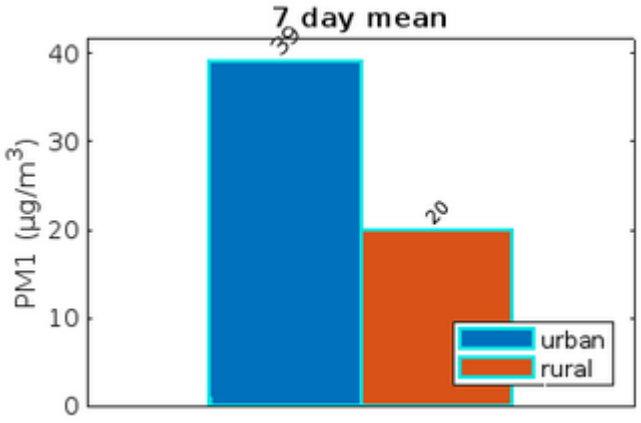


Figure 48 (source in Appendix I): MATLAB Visualization in ThingSpeak of PM1 7-day period

As we can see from the values and graphs, during the 7-day period the mean value of PM10 and PM2.5 for the urban area was **46 µg/m^3** and **41 µg/m^3** accordingly which classifies concentration as "Low" according to Chapter's 1.2 Figure. The rural area mean values for the same 7-day period were **28 µg/m^3** and **33 µg/m^3** which classifies the concentration as "Low" to "Very Low". The sensor we used also provided values for PM1 which are low (**39 µg/m^3** for urban and **20 µg/m^3** for rural) even though we don't have a reference chart to categorize the concentration, so we follow the general rule which is "the lower concentration of particulate matter, the better air quality".

The results about the urban area and the presence of particulate matter in the air are showing us that the air is generally clean of particles. We still see a difference in urban and rural area, with rural air being cleaner and healthier.

The following graphs and statistics are about the Sound level:

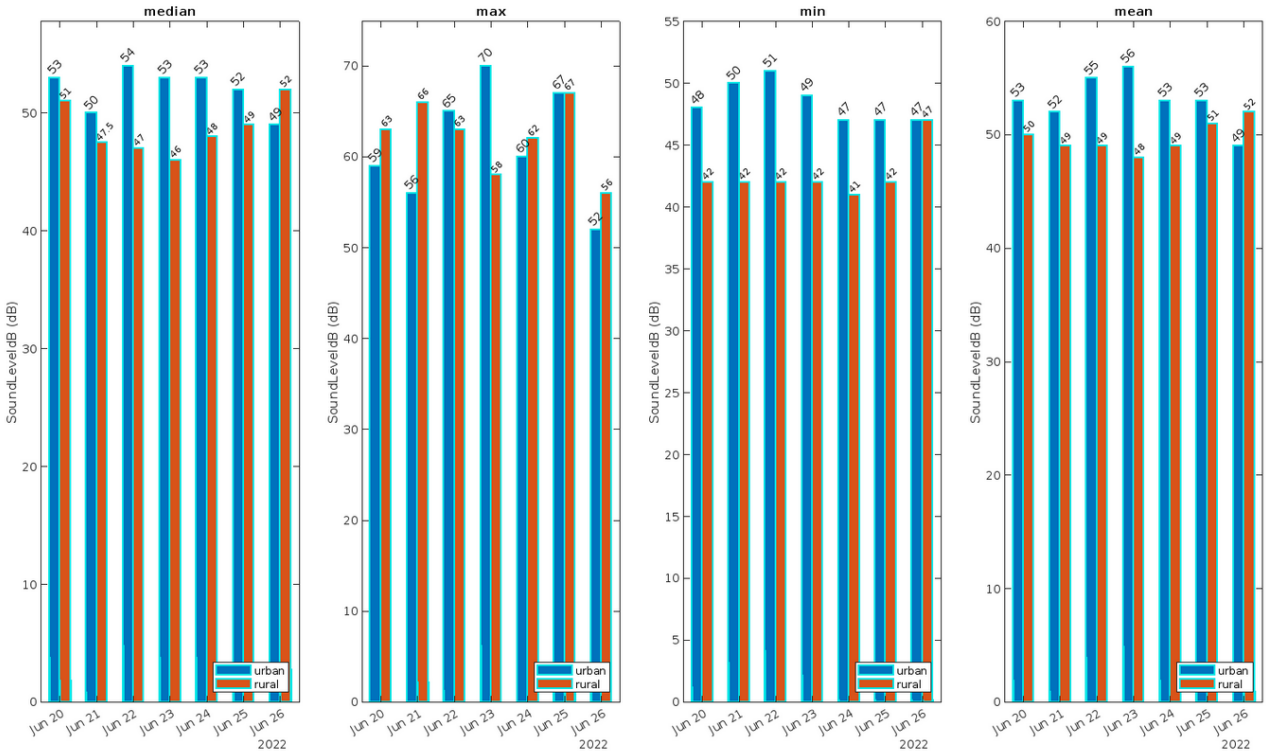


Figure 49 (source in Appendix I): MATLAB Visualization in ThingSpeak of Sound levels

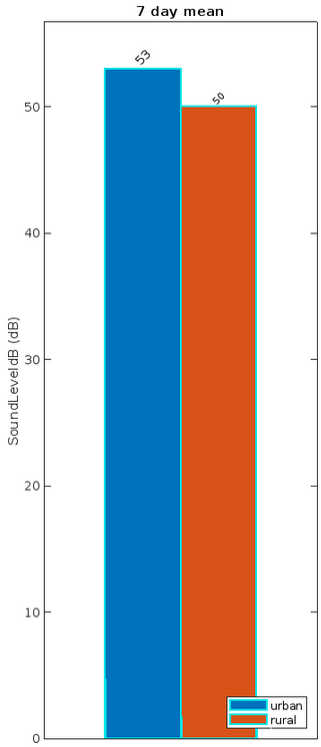


Figure 50 (source in Appendix I): MATLAB Visualization in ThingSpeak of Sound levels 7-day period

As we can see from the values and graphs, during the 7-day period the mean value of Sound level for the urban area was **53 dB** which classifies sound pollution as “Faint” to “Moderate to Quiet” according to Chapter’s 1.3 Figure. The rural area mean values for the same 7-day period were **50 dB** which classifies the sound pollution in the same category as the urban one.

The results about the urban area show that the territory where we placed the sensor to gather data is quiet with minimum sound pollution. Obviously since we only gather measurements from one place, they do not represent the sound pollution situation of the city very well. It is one of the improvements we recommend in the next chapter of the thesis, gathering data from multiple places. We can also observe that there is small difference (even though it’s logarithmic scale) between the urban and rural area.

7

Conclusion

Based on the findings and the data of the previous chapter, we can certainly conclude that the urban air contains elements that are hazardous for our health and the sound levels are elevated. Taking the extra step to gather further data from a rural area, provides us with the ability to compare the two areas: there is a clear difference in the indicators that we chose between an urban and a rural environment, the rural being much healthier and better for residents.

The initial goal of this thesis was accomplished: a robust implementation of sensor data gathering, and transmitting was achieved, and the data are (and will be) available for viewing and further analysis using the latest technologies and best tools available. As always, improvements can be made but this implementation can be the basis or inspiration of something better and larger.

7.1 Improvements

As mentioned earlier, the end node was placed for 7 days in 2 different areas, a rural and an urban. This gave us the opportunity to further analyze the data and make comparison between them.

Further improvements could be made in our implementation regarding the end nodes, such as adding different gases sensors for CO (Carbon monoxide), NO2 (Nitrogen dioxide), NO (Nitrogen monoxide), O3 (Ozone), SO2 (Sulfur dioxide), all of them being hazardous for our respiratory system if present in the environment of our residence. Having data for more air quality indicators provides a more holistic approach to quality monitoring and further actions and measures can be taken, specifically for every indicator. In addition, the energy consumption could be improved by enabling any deep-sleep function that is supported or by using different microcontrollers which are less energy consuming.

Another improvement that would directly impact the number of readings and therefore the output of our measurements is scaling. More end nodes equipped with sensors could be added in the urban area under monitoring and more gateways to support them and expand the coverage. Gathering measurements from different places in an area makes the statistics of the data more accurate and any models deriving from them, closer to the real situation that the area is facing.

Regarding the software, programming the ESP32 was done using C/C++. An improvement would be to use MicroPython which is a minified and very light version of Python, especially developed for microcontrollers. This would make the whole more modern but the challenge that someone would face would be interfacing with the sensors since there aren't many libraries yet for all sensors.

This page intentionally left blank

Bibliography - References

[1] https://www.researchgate.net/publication/331242686_Health_Effects_of_Air_Pollution_in_Urban_Environment

[2] https://www.euro.who.int/en/health-topics/environment-and-health/noise/noise

[3] https://en.wikipedia.org/wiki/Noise_pollution

[4] http://pdacontrolen.com/getting-started-considerations-and-concepts-lorawan-1/

[5] https://www.thethingsnetwork.org/docs/lorawan/what-is-lorawan/

[6] https://lora-alliance.org/resource_hub/lorawan-specification-v1-0-3/

[7] https://www.thethingsnetwork.org/docs/lorawan/architecture/

[8] https://www.thethingsnetwork.org/docs/lorawan/what-is-lorawan/

[9] https://en.wikipedia.org/wiki/ESP32

[10] https://cdn-shop.adafruit.com/datasheets/SSD1306.pdf

[11] http://www.dorji.com/docs/data/DRF1276G.pdf

[12] https://cdn-shop.adafruit.com/product-files/3179/sx1276_77_78_79.pdf

[13] https://www.semtech.com/products/wireless-rf/lora-core/sx1276

[14] https://www.sciosense.com/wp-content/uploads/documents/SC-001232-DS-2-CCS811B-Datasheet-Revision-2.pdf

[15] https://grobotronics.com/adafruit-ccs811-air-quality-sensor-breakout-voc-and-eco2.html

[16] https://www.ti.com/lit/gpn/hdc1080

[17] https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bmp280-ds001.pdf

[18] https://cityos-air.readme.io/docs/pms1003-digital-laser-dust-sensor-pm25-and-pm10

[19] https://invensense.tdk.com/products/digital/inmp441/

[20] https://docs.rakwireless.com/Product-Categories/WisLink/RAK831/Datasheet/

[21] https://en.wikipedia.org/wiki/ThingSpeak

Appendix I: List of Figures

Appendix II: LoRa end node source code

LoRa end node source code:

```
/*************************************************************************

   ESP32 : microcontroller

   SSD1306 128x64 OLED screen

   DRF1276G : LoRa transmitter

   CJMCU-8128 : air quality sensor
    TVCO (Total Volatile Organic Compound) (measured by CCS811)
    eCO2 (equivalent CO2) (measured by CCS811)
    Humidity (measured by SI702x)
    Temperature (measured by BMP280 and SI702x)
    Pressure (measured by BMP280)

   PMSA003 (PMS5003) : particle concentration/air quality sensor
    PM1,0 (less than 1Î¼m )
    PM2.5 (less than 2.5Î¼m )
    PM10 (less than 10Î¼m )

   INMP441 : omnidirectional MEMS microphone/sound level sensor



*************************************************************************/

// ================================================================================
======================| LIBRARIES
#include "SSD1306Ascii.h"
#include "SSD1306AsciiWire.h"

#include "Adafruit_CCS811.h"
#include "Adafruit_Si7021.h"
#include "Adafruit_BMP280.h"

#include <Wire.h>

#include <SoftwareSerial.h>

#include <TTN_esp32.h>
#include <TTN_CayenneLPP.h>

#include <driver/i2s.h>
#include "sos-iir-filter.h"
// ================================================================================
======================| VARIABLES_OBJECTS SENSORS, SCREEN
Adafruit_CCS811 ccs811;
Adafruit_BMP280 bmp280;                 // I2C
Adafruit_Si7021 SI702x = Adafruit_Si7021();

SSD1306AsciiWire oled;

SoftwareSerial pmsSerial(26, 25); //RX,TX

struct pms5003data {
  uint16_t framelen;
  uint16_t pm10_standard, pm25_standard, pm100_standard;
  uint16_t pm10_env, pm25_env, pm100_env;
  uint16_t particles_03um, particles_05um, particles_10um, particles_25um, particles_50um,
particles_100um;
  uint16_t unused;
  uint16_t checksum;
};
```

```
struct pms5003data data;

// ================================================================================
======================| VARIABLES_OBJECTS TTN
const char* devEui = "XXXXXXXXXXXXXXXXXXXX"; // Change to TTN Device EUI
//const char* devEui = "XXXXXXXXXXXXXXXXXXXX"; // Change to TTN Device EUI
const char* appEui = "XXXXXXXXXXXXXXXXXXXX"; // Change to TTN Application EUI
const char* appKey = "XXXXXXXXXXXXXXXXXXXX"; // Change to TTN Application Key
//const char* appKey = "XXXXXXXXXXXXXXXXXXXX"; // Change to TTN Application Key
TTN_esp32 ttn ;
TTN_CayenneLPP lpp;



// ================================================================================
======================| START microphone stuff
//
// Configuration
//

#define LEQ_PERIOD        1           // second(s)
#define WEIGHTING         C_weighting // Also avaliable: 'C_weighting' or 'None' (Z_weighti
ng)
#define LEQ_UNITS         "LAeq"      // customize based on above weighting used
#define DB_UNITS          "dBA"       // customize based on above weighting used


// NOTE: Some microphones require at least DC-Blocker filter
#define MIC_EQUALIZER     INMP441    // See below for defined IIR filters or set to 'None'
to disable
#define MIC_OFFSET_DB     3.0103      // Default offset (sine-wave RMS vs. dBFS). Modify th
is value for linear calibration

// Customize these values from microphone datasheet
#define MIC_SENSITIVITY   -26         // dBFS value expected at MIC_REF_DB (Sensitivity val
ue from datasheet)
#define MIC_REF_DB        94.0        // Value at which point sensitivity is specified in d
atasheet (dB)
#define MIC_OVERLOAD_DB   120.0       // dB - Acoustic overload point
#define MIC_NOISE_DB      -87          // dB - Noise floor
#define MIC_BITS          24          // valid number of bits in I2S data
#define MIC_CONVERT(s)    (s >> (SAMPLE_BITS - MIC_BITS))
#define MIC_TIMING_SHIFT  0           // Set to one to fix MSB timing for some microphones,
 i.e. SPH0645LM4H-x

// Calculate reference amplitude value at compile time
constexpr double MIC_REF_AMPL = pow(10, double(MIC_SENSITIVITY) / 20) * ((1 << (MIC_BITS -
1)) - 1);

//
// I2S pins - Can be routed to almost any (unused) ESP32 pin.
//            SD can be any pin, inlcuding input only pins (36-39).
//            SCK (i.e. BCLK) and WS (i.e. L/R CLK) must be output capable pins
//
// Below ones are just example for my board layout, put here the pins you will use
//
#define I2S_WS            15
#define I2S_SCK           12
#define I2S_SD            32

// I2S peripheral to use (0 or 1)
#define I2S_PORT          I2S_NUM_0



//
```

```
// IIR Filters
//

// DC-Blocker filter - removes DC component from I2S data
// See: https://www.dsprelated.com/freebooks/filters/DC_Blocker.html
// a1 = -0.9992 should heavily attenuate frequencies below 10Hz
SOS_IIR_Filter DC_BLOCKER = {
  gain: 1.0,
sos: {{ -1.0, 0.0, +0.9992, 0}}
};

//
// Equalizer IIR filters to flatten microphone frequency response
// See respective .m file for filter design. Fs = 48Khz.
//
// Filters are represented as Second-Order Sections cascade with assumption
// that b0 and a0 are equal to 1.0 and 'gain' is applied at the last step
// B and A coefficients were transformed with GNU Octave:
// [sos, gain] = tf2sos(B, A)
// See: https://www.dsprelated.com/freebooks/filters/Series_Second_Order_Sections.html
// NOTE: SOS matrix 'a1' and 'a2' coefficients are negatives of tf2sos output
//

// TDK/InvenSense ICS-43434
// Datasheet: https://www.invensense.com/wp-content/uploads/2016/02/DS-000069-ICS-43434-v1.
1.pdf
// B = [0.477326418836803, -0.486486982406126, -0.336455844522277, 0.234624646917202, 0.111
023257388606];
// A = [1.0, -1.93073383849136326, 0.86519456089576796, 0.06442838283825100, 0.001112492988
00616];
SOS_IIR_Filter ICS43434 = {
  gain: 0.477326418836803,
sos: { // Second-Order Sections {b1, b2, -a1, -a2}
    { +0.96986791463971267, 0.23515976355743193, -0.06681948004769928, -0.00111521990688128
},
    { -1.98905931743624453, 0.98908924206960169, +1.99755331853906037, -0.99755481510122113
}
  }
};

// TDK/InvenSense ICS-43432
// Datasheet: https://www.invensense.com/wp-content/uploads/2015/02/ICS-43432-data-sheet-v1
.3.pdf
// B = [-0.457337023383413   1.12228667105574775  -0.77818278904413563, 0.009689263379780
37, 0.10345668405223755]
// A = [1.0, -3.3420781082912949, 4.4033694320978771, -3.0167072679918010, 1.22655365676470
31, -0.2962229189311990, 0.0251085747458112]
SOS_IIR_Filter ICS43432 = {
gain: 0.457337023383413,
sos: { // Second-Order Sections {b1, b2, -a1, -a2}
    { -0.544047931916859, -0.248361759321800, +0.403298891662298, -0.207346186351843},
    { -1.909911869441421, +0.910830292683527, +1.790285722826743, -0.804085812369134},
    { +0.000000000000000, +0.000000000000000, +1.148493493802252, -0.150599527756651}
  }
};

// TDK/InvenSense INMP441
// Datasheet: https://www.invensense.com/wp-content/uploads/2015/02/INMP441.pdf
// B ~= [1.00198, -1.99085, 0.98892]
// A ~= [1.0, -1.99518, 0.99518]
SOS_IIR_Filter INMP441 = {
  gain: 1.00197834654696,
sos: { // Second-Order Sections {b1, b2, -a1, -a2}
    { -1.986920458344451, +0.986963226946616, +1.995178510504166, -0.995184322194091}
  }
};

// Infineon IM69D130 Shield2Go
// Datasheet: https://www.infineon.com/dgdl/Infineon-IM69D130-DS-v01_00-EN.pdf?fileId=5546d
```

```
462602a9dc801607a0e46511a2e
// B ~= [1.001240684967527, -1.996936108836337, 0.995703101823006]
// A ~= [1.0, -1.997675693595542, 0.997677044195563]
// With additional DC blocking component
SOS_IIR_Filter IM69D130 = {
  gain: 1.00124068496753,
sos: {
    { -1.0, 0.0, +0.9992, 0}, // DC blocker, a1 = -0.9992
    { -1.994461610298131, 0.994469278738208, +1.997675693595542, -0.997677044195563}
  }
};

// Knowles SPH0645LM4H-B, rev. B
// https://cdn-shop.adafruit.com/product-files/3421/i2S+Datasheet.PDF
// B ~= [1.001234, -1.991352, 0.990149]
// A ~= [1.0, -1.993853, 0.993863]
// With additional DC blocking component
SOS_IIR_Filter SPH0645LM4H_B_RB = {
  gain: 1.00123377961525,
sos: { // Second-Order Sections {b1, b2, -a1, -a2}
    { -1.0, 0.0, +0.9992, 0}, // DC blocker, a1 = -0.9992
    { -1.988897663539382, +0.988928479008099, +1.993853376183491, -0.993862821429572}
  }
};

//
// Weighting filters
//

//
// A-weighting IIR Filter, Fs = 48KHz
// (By Dr. Matt L., Source: https://dsp.stackexchange.com/a/36122)
// B = [0.169994948147430, 0.280415310498794, -1.120574766348363, 0.131562559965936, 0.9741
53561246036, -0.282740857326553, -0.152810756202003]
// A = [1.0, -2.12979364760736134, 0.42996125885751674, 1.62132698199721426, -0.96669962900
852902, 0.00121015844426781, 0.04400300696788968]
SOS_IIR_Filter A_weighting = {
  gain: 0.169994948147430,
sos: { // Second-Order Sections {b1, b2, -a1, -a2}
    { -2.00026996133106, +1.00027056142719, -1.060868438509278, -0.163987445885926},
    { +4.35912384203144, +3.09120265783884, +1.208419926363593, -0.273166998428332},
    { -0.70930303489759, -0.29071868393580, +1.982242159753048, -0.982298594928989}
  }
};

//
// C-weighting IIR Filter, Fs = 48KHz
// Designed by invfreqz curve-fitting, see respective .m file
// B = [-0.49164716933714026, 0.14844753846498662, 0.74117815661529129, -0.0328187833403931
4, -0.29709276192593875, -0.06442545322197900, -0.00364152725482682]
// A = [1.0, -1.0325358998928318, -0.9524000181023488, 0.8936404694728326   0.2256286147169
398  -0.1499917107550188, 0.0156718181681081]
SOS_IIR_Filter C_weighting = {
gain: -0.491647169337140,
sos: {
    { +1.4604385758204708, +0.5275070373815286, +1.9946144559930252, -0.9946217070140883},
    { +0.2376222404939509, +0.0140411206016894, -1.3396585608422749, -0.4421457807694559},
    { -2.0000000000000000, +1.0000000000000000, +0.3775800047420818, -0.0356365756680430}
  }
};


//
// Sampling
//
#define SAMPLE_RATE       48000 // Hz, fixed to design of IIR filters
#define SAMPLE_BITS       32    // bits
#define SAMPLE_T          int32_t
#define SAMPLES_SHORT     (SAMPLE_RATE / 8) // ~125ms
```

```
#define SAMPLES_LEQ       (SAMPLE_RATE * LEQ_PERIOD)
#define DMA_BANK_SIZE     (SAMPLES_SHORT / 16)
#define DMA_BANKS         32

// Data we push to 'samples_queue'
struct sum_queue_t {
  // Sum of squares of mic samples, after Equalizer filter
  float sum_sqr_SPL;
  // Sum of squares of weighted mic samples
  float sum_sqr_weighted;
  // Debug only, FreeRTOS ticks we spent processing the I2S data
  uint32_t proc_ticks;
};
QueueHandle_t samples_queue;

// Static buffer for block of samples
float samples[SAMPLES_SHORT] __attribute__((aligned(4)));

//
// I2S Microphone sampling setup
//
void mic_i2s_init() {
  // Setup I2S to sample mono channel for SAMPLE_RATE * SAMPLE_BITS
  // NOTE: Recent update to Arduino_esp32 (1.0.2 -> 1.0.3)
  //       seems to have swapped ONLY_LEFT and ONLY_RIGHT channels
  const i2s_config_t i2s_config = {
mode: i2s_mode_t(I2S_MODE_MASTER | I2S_MODE_RX),
sample_rate: SAMPLE_RATE,
bits_per_sample: i2s_bits_per_sample_t(SAMPLE_BITS),
channel_format: I2S_CHANNEL_FMT_ONLY_LEFT,
communication_format: i2s_comm_format_t(I2S_COMM_FORMAT_I2S | I2S_COMM_FORMAT_I2S_MSB),
intr_alloc_flags: ESP_INTR_FLAG_LEVEL1,
dma_buf_count: DMA_BANKS,
dma_buf_len: DMA_BANK_SIZE,
use_apll: true,
tx_desc_auto_clear: false,
    fixed_mclk: 0
  };
  // I2S pin mapping
  const i2s_pin_config_t pin_config = {
bck_io_num:   I2S_SCK,
ws_io_num:    I2S_WS,
data_out_num: -1, // not used
data_in_num:  I2S_SD
  };

  i2s_driver_install(I2S_PORT, &i2s_config, 0, NULL);

#if (MIC_TIMING_SHIFT > 0)
  // Undocumented (?!) manipulation of I2S peripheral registers
  // to fix MSB timing issues with some I2S microphones
  REG_SET_BIT(I2S_TIMING_REG(I2S_PORT), BIT(9));
  REG_SET_BIT(I2S_CONF_REG(I2S_PORT), I2S_RX_MSB_SHIFT);
#endif

  i2s_set_pin(I2S_PORT, &pin_config);

  //FIXME: There is a known issue with esp-idf and sampling rates, see:
  //       https://github.com/espressif/esp-idf/issues/2634
  //       In the meantime, the below line seems to set sampling rate at ~47999.992Hz
  //       fifs_req=24576000, sdm0=149, sdm1=212, sdm2=5, odir=2 -> fifs_reached=24575996
  //NOTE:  This seems to be fixed in ESP32 Arduino 1.0.4, esp-idf 3.2
  //       Should be safe to remove...
  //#include <soc/rtc.h>
  //rtc_clk_apll_enable(1, 149, 212, 5, 2);
}

//
// I2S Reader Task
```

```
//
// Rationale for separate task reading I2S is that IIR filter
// processing cam be scheduled to different core on the ESP32
// while main task can do something else, like update the
// display in the example
//
// As this is intended to run as separate hihg-priority task,
// we only do the minimum required work with the I2S data
// until it is 'compressed' into sum of squares
//
// FreeRTOS priority and stack size (in 32-bit words)
#define I2S_TASK_PRI   4
#define I2S_TASK_STACK 2048
//
void mic_i2s_reader_task(void* parameter) {
  mic_i2s_init();

  // Discard first block, microphone may have startup time (i.e. INMP441 up to 83ms)
  size_t bytes_read = 0;
  i2s_read(I2S_PORT, &samples, SAMPLES_SHORT * sizeof(int32_t), &bytes_read, portMAX_DELAY)
;

  while (true) {
    // Block and wait for microphone values from I2S
    //
    // Data is moved from DMA buffers to our 'samples' buffer by the driver ISR
    // and when there is requested ammount of data, task is unblocked
    //
    // Note: i2s_read does not care it is writing in float[] buffer, it will write
    //       integer values to the given address, as received from the hardware peripheral.
    i2s_read(I2S_PORT, &samples, SAMPLES_SHORT * sizeof(SAMPLE_T), &bytes_read, portMAX_DEL
AY);

    TickType_t start_tick = xTaskGetTickCount();

    // Convert (including shifting) integer microphone values to floats,
    // using the same buffer (assumed sample size is same as size of float),
    // to save a bit of memory
    SAMPLE_T* int_samples = (SAMPLE_T*)&samples;
    for (int i = 0; i < SAMPLES_SHORT; i++) samples[i] = MIC_CONVERT(int_samples[i]);

    sum_queue_t q;
    // Apply equalization and calculate Z-weighted sum of squares,
    // writes filtered samples back to the same buffer.
    q.sum_sqr_SPL = MIC_EQUALIZER.filter(samples, samples, SAMPLES_SHORT);

    // Apply weighting and calucate weigthed sum of squares
    q.sum_sqr_weighted = WEIGHTING.filter(samples, samples, SAMPLES_SHORT);

    // Debug only. Ticks we spent filtering and summing block of I2S data
    q.proc_ticks = xTaskGetTickCount() - start_tick;

    // Send the sums to FreeRTOS queue where main task will pick them up
    // and further calcualte decibel values (division, logarithms, etc...)
    xQueueSend(samples_queue, &q, portMAX_DELAY);
  }
}


// =============================================================================
======================| END microphone stuff
```

```
// =================================================================================
======================| SETUP
void setup() {
  delay(5000);
  Serial.begin(115200);

  Serial.println("\t\t\tSTART OF SETUP");

  LMIC_setClockError(MAX_CLOCK_ERROR * 10 / 100);

  // Enable I2C
  Wire.begin(21, 22); // SDA, SCL
  Wire.setClock(400000L);

  // sensor baud rate is 9600
  pmsSerial.begin(9600);


  /* --- SETUP screen ------ */
  oled.begin(&Adafruit128x64, 0x3C);
  oled.setFont(System5x7);
  oled.setScrollMode(SCROLL_MODE_AUTO);
  oled.clear();
  oled.println("Hello world!");

  /* --- SETUP ccs811 - air quality module ------ */
  Serial.println("CCS811 test");      /* --- SETUP CCS811 on 0x5A ------ */
  //  ccs811.set_i2cdelay(50); // Needed for ESP8266 because it doesn't handle I2C clock st
retch correctly
  if (!ccs811.begin()) {
    Serial.println("Failed to start sensor! Please check your wiring.");
    while (true);
  }

  /* --- SETUP bmp280 - barometric pressure/temperature ------ */
  Serial.println("BMP280 test");     /* --- SETUP BMP on 0x76 ------ */
  if (!bmp280.begin(0x76)) {
    Serial.println("Could not find a valid BMP280 sensor, check wiring!");
    while (true);
  }

  /* --- SETUP si7021 - temperature/humidity ------ */
  Serial.println("Si7021 test!");     /* ---- SETUP SI702x ----- */
  if (!SI702x.begin()) {
    Serial.println("Did not find Si702x sensor!");
    while (true);
  }
  Serial.print("Found model ");
  switch (SI702x.getModel()) {
    case SI_Engineering_Samples:
      Serial.print("SI engineering samples"); break;
    case SI_7013:
      Serial.print("Si7013"); break;
    case SI_7020:
      Serial.print("Si7020"); break;
    case SI_7021:
      Serial.print("Si7021"); break;
    case SI_UNKNOWN:
    default:
      Serial.print("Unknown");
  }
  Serial.print(" Revision(");
```

```
  Serial.print(SI702x.getRevision());
  Serial.print(")");
  Serial.print(" Serial #"); Serial.print(SI702x.sernum_a, HEX); Serial.println(SI702x.sern
um_b, HEX);

  Serial.println("---");
  /* --- SETUP drf1276g ------ */
  // 18, 19, 23, 5
  // CLK,MISO,MOSI,SS
  ttn.begin(/*nss*/5, /*rxtx*/ -1, /*rst*/14, /*dio0*/2, /*dio1*/34, /*dio2*/35);


  ttn.onMessage(message); // Declare callback function for handling downlink
  // messages from server

  ttn.join(devEui, appEui, appKey);

  Serial.print("Joining TTN ");
  while (!ttn.isJoined())
  {
    Serial.print(".");
    delay(1000);
  }
  Serial.println("\nJOINED !");
  ttn.showStatus();
  Serial.print("Port: "); Serial.println(ttn.getPort());
  Serial.println("\t\t\tEND OF SETUP\n\n");




  delay(5000);
}
// ================================================================================
======================| LOOP
void loop() {
  Serial.println("\n\n\t\t\tSTART OF LOOP");

  lpp.reset();

  Serial.println("\tBMP280");
  Serial.print("Temperature = ");  Serial.println(bmp280.readTemperature());
  oled.println(bmp280.readTemperature());
  Serial.print("Pressure = ");  Serial.println(bmp280.readPressure() / 100);
  oled.println(bmp280.readPressure() / 100);
  //Serial.println(" Pa, ");

  Serial.println("\tSI7021");
  Serial.print("Temperature = ");  Serial.println(SI702x.readTemperature(), 2);
  oled.println(SI702x.readTemperature(), 2);
  //Serial.print(" Â°C, ");
  Serial.print("Humidity = ");  Serial.println(SI702x.readHumidity(), 2);
  oled.println(SI702x.readHumidity(), 2);



  uint16_t eco2, etvoc, errstat, raw;     // Read CCS811
  if (ccs811.available()) {
    if (!ccs811.readData()) {
      Serial.println("\tCCS811");
      Serial.print("CO2: ");      Serial.println(ccs811.geteCO2());
      oled.println(ccs811.geteCO2());
      //Serial.println("ppm");
      Serial.print("TVOC: ");      Serial.println(ccs811.getTVOC());
      oled.println(ccs811.getTVOC());
    }
  }
  oled.println("--------");
```

```
  //-------------------------------------------------------------------------------------
-------------------------|


  // Create FreeRTOS queue
  samples_queue = xQueueCreate(8, sizeof(sum_queue_t));

  // Create the I2S reader FreeRTOS task
  // NOTE: Current version of ESP-IDF will pin the task
  //       automatically to the first core it happens to run on
  //       (due to using the hardware FPU instructions).
  //       For manual control see: xTaskCreatePinnedToCore
  xTaskCreate(mic_i2s_reader_task, "Mic I2S Reader", I2S_TASK_STACK, NULL, I2S_TASK_PRI, NU
LL);

  sum_queue_t q;
  uint32_t Leq_samples = 0;
  double Leq_sum_sqr = 0;
  double Leq_dB = 0;


  // Read sum of samaples, calculated by 'i2s_reader_task'
  while (xQueueReceive(samples_queue, &q, portMAX_DELAY)) {

    // Calculate dB values relative to MIC_REF_AMPL and adjust for microphone reference
    double short_RMS = sqrt(double(q.sum_sqr_SPL) / SAMPLES_SHORT);
    double short_SPL_dB = MIC_OFFSET_DB + MIC_REF_DB + 20 * log10(short_RMS / MIC_REF_AMPL)
;

    // In case of acoustic overload or below noise floor measurement, report infinty Leq va
lue
    if (short_SPL_dB > MIC_OVERLOAD_DB) {
      Leq_sum_sqr = INFINITY;
    } else if (isnan(short_SPL_dB) || (short_SPL_dB < MIC_NOISE_DB)) {
      Leq_sum_sqr = -INFINITY;
    }

    // Accumulate Leq sum
    Leq_sum_sqr += q.sum_sqr_weighted;
    Leq_samples += SAMPLES_SHORT;

    // When we gather enough samples, calculate new Leq value
    if (Leq_samples >= SAMPLE_RATE * LEQ_PERIOD) {
      double Leq_RMS = sqrt(Leq_sum_sqr / Leq_samples);
      Leq_dB = MIC_OFFSET_DB + MIC_REF_DB + 20 * log10(Leq_RMS / MIC_REF_AMPL);
      Leq_sum_sqr = 0;
      Leq_samples = 0;

      // Serial output, customize (or remove) as needed
      //Serial.printf("%.1f\n", Leq_dB);
      //delay(1000);
      // Debug only
      //Serial.printf("%u processing ticks\n", q.proc_ticks);
      break;
    }


  }
```

```
  //-------------------------------------------------------------------------------------------
-------------------------|


  //-------------------------------------------------------------------------------------------
-------------------------|
  if (readPMSdata(&pmsSerial)) {
    // reading data was successful!

    Serial.println("---------------------------------------");

    Serial.println("\tPMSA003");
    Serial.println("Concentration Units (environmental)");
    Serial.print("PM 1.0: "); Serial.println(data.pm10_env);
    Serial.print("PM 2.5: "); Serial.println(data.pm25_env);
    Serial.print("PM 10: "); Serial.println(data.pm100_env);
    Serial.println("---------------------------------------");

  }

  Serial.println("\tINMP441");
  Serial.print("Sound level: "); Serial.println(int(round(Leq_dB)));
  //-------------------------------------------------------------------------------------------
-------------------------| START: prepare and send data to TTN




  lpp.addDigitalOutput(1, int(round(bmp280.readTemperature())));
  lpp.addDigitalOutput(2, int(round(bmp280.readPressure() / 100)));
  lpp.addDigitalOutput(3, int(round(SI702x.readHumidity())));
  //  lpp.addDigitalOutput(4, int(round(ccs811.geteCO2())));
  //  lpp.addDigitalOutput(5, int(round(ccs811.getTVOC())));
  //  lpp.addDigitalOutput(6, int(round(data.pm10_env)));
  //  lpp.addDigitalOutput(7, int(round(data.pm25_env)));
  //  lpp.addDigitalOutput(8, int(round(data.pm100_env)));
  //  lpp.addDigitalOutput(9, int(round(Leq_dB)));




  Serial.println("---");

  Serial.print("lpp buffer in TTN will be: ");
  for (int i = 0; i < lpp.getSize(); i++)
  {
    Serial.printf(" %02X", lpp.getBuffer()[i]);

  }

  Serial.println();


  Serial.print("lpp buffer in STRING will be (channel port value): ");
  for (int i = 0; i < lpp.getSize(); i++)
  {
    Serial.print(" " + String(lpp.getBuffer()[i]));

  }

  Serial.println();
```

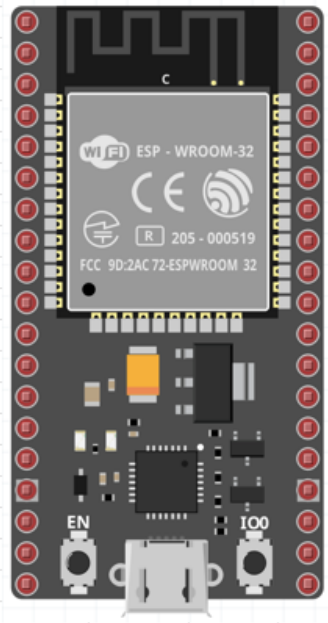
```
  Serial.println("---");


  // bool sendBytes(uint8_t* payload, size_t length, uint8_t port = 1, uint8_t confirm = 0)
;

  ttn.sendBytes(lpp.getBuffer(), lpp.getSize());

  //  if (ttn.sendBytes(lpp.getBuffer(), lpp.getSize()))
  //  {
  //    Serial.printf("Temp: %f TTN_CayenneLPP: %d %x %02X%02X\n", round(bmp280.readTempera
ture()), lpp.getBuffer()[0], lpp.getBuffer()[1],
  //                  lpp.getBuffer()[2], lpp.getBuffer()[3]);
  //  } else {
  //    Serial.println("ERROR while sending to TTN...");
  //  }

  //-------------------------------------------------------------------------------------
-----------------------| END: prepare and send data to TTN


  Serial.println("\n\n\t\t\tEND OF LOOP");
  delay(30000);
}
// =====================================================================================
======================| FUNCTIONS

// TTN related
void message(const uint8_t* payload, size_t size, int rssi)
{
  Serial.println("-- MESSAGE");
  Serial.print("Received " + String(size) + " bytes RSSI=" + String(rssi) + "db");
  for (int i = 0; i < size; i++)
  {
    Serial.print(" " + String(payload[i]));
    // Serial.write(payload[i]);
  }
}



// PMSA003 related
boolean readPMSdata(Stream *s) {
  if (! s->available()) {
    return false;
  }

  // Read a byte at a time until we get to the special '0x42' start-byte
  if (s->peek() != 0x42) {
    s->read();
    return false;
  }

  // Now read all 32 bytes
  if (s->available() < 32) {
    return false;
  }

  uint8_t buffer[32];
  uint16_t sum = 0;
  s->readBytes(buffer, 32);

  // get checksum ready
  for (uint8_t i = 0; i < 30; i++) {
    sum += buffer[i];
  }
```

```
  // The data comes in endian'd, this solves it so it works on all platforms
  uint16_t buffer_u16[15];
  for (uint8_t i = 0; i < 15; i++) {
    buffer_u16[i] = buffer[2 + i * 2 + 1];
    buffer_u16[i] += (buffer[2 + i * 2] << 8);
  }

  // put it into a nice struct :)
  memcpy((void *)&data, (void *)buffer_u16, 30);

  if (sum != data.checksum) {
    Serial.println("Checksum failure");
    return false;
  }
  // success!
  return true;
}
```

Appendix III: LoRa end node and sensors pin connections

LoRa end node and sensors connection diagram:

Sensor	Sensor pin	ESP32 pin			ESP32 pin	Sensor pin	Sensor
ALL	VBAT , VDD , VCC	3V3	1		1	GND	GND , WAK	ALL , CJMCU8128
		EN	2		2	23	MOSI	DRF1276G
		SP	3		3	22	SCL	CJMCU8128 , SSD1306
		SN	4		4	TX		
DRF1276G	DIO1	34	5		5	RX		
DRF1276G	DIO2	35	6		6	21	SDA	CJMCU8128 , SSD1306
INMP441	SD	32	7		7	GND		
		33	8		8	19	MISO	DRF1276G
PMSA003	TX	25	9		9	18	SCK	DRF1276G
PMSA003	RX	26	10		10	5	NSS	DRF1276G
		27	11		11	17		
DRF1276G	NRST	14	12		12	16		
INMP441	SCK	12	13		13	4		
		GND	14		14	0		
INMP441	WS	13	15		15	2	DIO0	DRF1276G
		SD2	16		16	15	WS	INMP441
		SD3	17		17	SD1		
		CMD	18		18	SD0		
		V5	19		19	CLK		



Appendix IV: MATLAB code for ThingSpeak

ThingSpeak MATLAB code for Visualization of values of all readings:

```
% Enter your MATLAB code below
% tiledlayout(rows,columns)
tiledlayout(4,2)
%.WindowState = 'maximized';
readChannelID = XXXXX;
myfield1 = 1;
myfield2 = 2;
myfield3 = 3;
myfield4 = 4;
myfield5 = 5;
myfield6 = 6;
myfield7 = 7;
myfield8 = 8;
readAPIKey = 'XXXXX';
[data, timeStamps ] = thingSpeakRead(readChannelID,...
    'Fields',[myfield1 myfield2 myfield3 myfield4 myfield5 myfield6 myfield7 myfield8 ],...
    'NumDays',7,...
    'ReadKey',readAPIKey);
mydata1 = data(:, 1);
mydata2 = data(:, 2);
mydata3 = data(:, 3);
mydata4 = data(:, 4);
mydata5 = data(:, 5);
mydata6 = data(:, 6);
mydata7 = data(:, 7);
mydata8 = data(:, 8);
%-----------------------------------------------| PLOT
nexttile
plot(timeStamps, mydata1,'Marker','square');
axis padded
xtickformat('dd/MM HH:mm')
xtickangle(45)
title('Temperature')
nexttile
plot(timeStamps, mydata2,'Marker','square');
axis padded
xtickformat('dd/MM HH:mm')
xtickangle(45)
title('Humidity')
nexttile
plot(timeStamps, mydata3,'Marker','square');
axis padded
xtickformat('dd/MM HH:mm')
xtickangle(45)
title('TVOC')
nexttile
plot(timeStamps, mydata4,'Marker','square');
axis padded
xtickformat('dd/MM HH:mm')
xtickangle(45)
title('eCO2')
nexttile
plot(timeStamps, mydata5,'Marker','square');
axis padded
xtickformat('dd/MM HH:mm')
xtickangle(45)
title('PM 10')
nexttile
plot(timeStamps, mydata6,'Marker','square');
axis padded
xtickformat('dd/MM HH:mm')
xtickangle(45)
title('PM 2.5')
nexttile
plot(timeStamps, mydata7,'Marker','square');
axis padded
xtickformat('dd/MM HH:mm')
xtickangle(45)
title('PM 1')
```

```
nexttile
plot(timeStamps, mydata8,'Marker','square');
axis padded
xtickformat('dd/MM HH:mm')
xtickangle(45)
title('Sound level dB')
```

ThingSpeak MATLAB code for Visualization of values of PM readings:

```
myid = XXXXX;
myfield = 1;
mykey = 'XXXXX';
%{
[mydata,mytimestamps] = thingSpeakRead(myid, ...
    'Fields',[1 2 3 4 5 6 7 8], ...
    'numDays',3, ...
    'ReadKey',mykey, ...
    OutputFormat='timetable')
%}
mydata_rural= thingSpeakRead(myid, ...
    'Fields',[1 2 3 4 5 6 7 8], ...
    'ReadKey',mykey, ...
    OutputFormat='timetable', ...
    DateRange=[datetime('2022-06-01'),datetime('2022-06-08')]);
mydata_urban= thingSpeakRead(myid, ...
    'Fields',[1 2 3 4 5 6 7 8], ...
    'ReadKey',mykey, ...
    OutputFormat='timetable', ...
    DateRange=[datetime('2022-06-20'),datetime('2022-06-27')]);
dailymax_rural = retime(mydata_rural, 'daily', 'max');
dailymin_rural = retime(mydata_rural, 'daily', 'min');
dailymean_rural = retime(mydata_rural, 'daily', 'mean');
dailymean7_rural = retime(mydata_rural, 'regular','mean', 'TimeStep',days(7));
dailymedian_rural = retime(mydata_rural, 'daily', 'median');
dailymax_urban = retime(mydata_urban, 'daily', 'max');
dailymin_urban = retime(mydata_urban, 'daily', 'min');
dailymean_urban = retime(mydata_urban, 'daily', 'mean');
dailymean7_urban = retime(mydata_urban, 'regular','mean', 'TimeStep',days(7));
dailymedian_urban = retime(mydata_urban, 'daily', 'median');
%-------------------------------------------------| PLOT
tiledlayout(3,5)
%-------------------------------------------------| PLOT - PM10
nexttile
bbl=bar(dailymedian_urban.Timestamps,...
    [dailymedian_urban.PM10 dailymedian_rural.PM10],...
    1,'EdgeColor',[0 .9 .9],'LineWidth',1.5);
axis padded
xtips1 = bbl(1).XEndPoints;
ytips1 = bbl(1).YEndPoints;
labels1 = string(bbl(1).YData);
text(xtips1,ytips1,labels1,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 10,'rotation',45)
xtips2 = bbl(2).XEndPoints;
ytips2 = bbl(2).YEndPoints;
labels2 = string(bbl(2).YData);
text(xtips2,ytips2,labels2,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 8,'rotation',45)
title('median')
ylabel('PM10 (Î¼g/m^3)')
legend({'urban','rural'},'Location','southeast')
%---
nexttile
bbl=bar(dailymax_urban.Timestamps,...
    [dailymax_urban.PM10 dailymax_rural.PM10],...
    1,'EdgeColor',[0 .9 .9],'LineWidth',1.5);
axis padded
xtips1 = bbl(1).XEndPoints;
ytips1 = bbl(1).YEndPoints;
labels1 = string(bbl(1).YData);
text(xtips1,ytips1,labels1,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 10,'rotation',45)
xtips2 = bbl(2).XEndPoints;
ytips2 = bbl(2).YEndPoints;
labels2 = string(bbl(2).YData);
text(xtips2,ytips2,labels2,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 8,'rotation',45)
title('max')
ylabel('PM10 (Î¼g/m^3)')
```

```
legend({'urban','rural'},'Location','southeast')
%---
nexttile
bbl=bar(dailymin_urban.Timestamps,...
    [dailymin_urban.PM10 dailymin_rural.PM10],...
    1,'EdgeColor',[0 .9 .9],'LineWidth',1.5);
axis padded
xtips1 = bbl(1).XEndPoints;
ytips1 = bbl(1).YEndPoints;
labels1 = string(bbl(1).YData);
text(xtips1,ytips1,labels1,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 10,'rotation',45)
xtips2 = bbl(2).XEndPoints;
ytips2 = bbl(2).YEndPoints;
labels2 = string(bbl(2).YData);
text(xtips2,ytips2,labels2,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 8,'rotation',45)
title('min')
ylabel('PM10 (Î¼g/m^3)')
legend({'urban','rural'},'Location','southeast')
%---
nexttile
bbl=bar(dailymean_urban.Timestamps,...
    round([dailymean_urban.PM10 dailymean_rural.PM10]),...
    1,'EdgeColor',[0 .9 .9],'LineWidth',1.5);
axis padded
xtips1 = bbl(1).XEndPoints;
ytips1 = bbl(1).YEndPoints;
labels1 = string(bbl(1).YData);
text(xtips1,ytips1,labels1,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 10,'rotation',45)
xtips2 = bbl(2).XEndPoints;
ytips2 = bbl(2).YEndPoints;
labels2 = string(bbl(2).YData);
text(xtips2,ytips2,labels2,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 8,'rotation',45)
title('mean')
ylabel('PM10 (Î¼g/m^3)')
legend({'urban','rural'},'Location','southeast')
%---
nexttile
bbl=bar(dailymean7_urban.Timestamps,...
    round([dailymean7_urban.PM10 dailymean7_rural.PM10]),...
    1,'EdgeColor',[0 .9 .9],'LineWidth',1.5);
axis padded
tickMarks = {'XTick',[]};
set(gca,tickMarks{:});
xtips1 = bbl(1).XEndPoints;
ytips1 = bbl(1).YEndPoints;
labels1 = string(bbl(1).YData);
text(xtips1,ytips1,labels1,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 10,'rotation',45)
xtips2 = bbl(2).XEndPoints;
ytips2 = bbl(2).YEndPoints;
labels2 = string(bbl(2).YData);
text(xtips2,ytips2,labels2,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 8,'rotation',45)
title('7 day mean')
ylabel('PM10 (Î¼g/m^3)')
legend({'urban','rural'},'Location','southeast')
%-------------------------------------------------| PLOT - PM25
nexttile
bbl=bar(dailymedian_urban.Timestamps,...
    [dailymedian_urban.PM25 dailymedian_rural.PM25],...
    1,'EdgeColor',[0 .9 .9],'LineWidth',1.5);
axis padded
xtips1 = bbl(1).XEndPoints;
ytips1 = bbl(1).YEndPoints;
labels1 = string(bbl(1).YData);
```

```
text(xtips1,ytips1,labels1,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 10,'rotation',45)
xtips2 = bb1(2).XEndPoints;
ytips2 = bb1(2).YEndPoints;
labels2 = string(bb1(2).YData);
text(xtips2,ytips2,labels2,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 8,'rotation',45)
title('median')
ylabel('PM25 (Î¼g/m^3)')
legend({'urban','rural'},'Location','southeast')
%---
nexttile
bb1=bar(dailymax_urban.Timestamps,...
    [dailymax_urban.PM25 dailymax_rural.PM25],...
    1,'EdgeColor',[0 .9 .9],'LineWidth',1.5);
axis padded
xtips1 = bb1(1).XEndPoints;
ytips1 = bb1(1).YEndPoints;
labels1 = string(bb1(1).YData);
text(xtips1,ytips1,labels1,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 10,'rotation',45)
xtips2 = bb1(2).XEndPoints;
ytips2 = bb1(2).YEndPoints;
labels2 = string(bb1(2).YData);
text(xtips2,ytips2,labels2,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 8,'rotation',45)
title('max')
ylabel('PM25 (Î¼g/m^3)')
legend({'urban','rural'},'Location','southeast')
%---
nexttile
bb1=bar(dailymin_urban.Timestamps,...
    [dailymin_urban.PM25 dailymin_rural.PM25],...
    1,'EdgeColor',[0 .9 .9],'LineWidth',1.5);
axis padded
xtips1 = bb1(1).XEndPoints;
ytips1 = bb1(1).YEndPoints;
labels1 = string(bb1(1).YData);
text(xtips1,ytips1,labels1,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 10,'rotation',45)
xtips2 = bb1(2).XEndPoints;
ytips2 = bb1(2).YEndPoints;
labels2 = string(bb1(2).YData);
text(xtips2,ytips2,labels2,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 8,'rotation',45)
title('min')
ylabel('PM25 (Î¼g/m^3)')
legend({'urban','rural'},'Location','southeast')
%---
nexttile
bb1=bar(dailymean_urban.Timestamps,...
    round([dailymean_urban.PM25 dailymean_rural.PM25]),...
    1,'EdgeColor',[0 .9 .9],'LineWidth',1.5);
axis padded
xtips1 = bb1(1).XEndPoints;
ytips1 = bb1(1).YEndPoints;
labels1 = string(bb1(1).YData);
text(xtips1,ytips1,labels1,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 10,'rotation',45)
xtips2 = bb1(2).XEndPoints;
ytips2 = bb1(2).YEndPoints;
labels2 = string(bb1(2).YData);
text(xtips2,ytips2,labels2,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 8,'rotation',45)
title('mean')
ylabel('PM25 (Î¼g/m^3)')
legend({'urban','rural'},'Location','southeast')
%---
nexttile
```

```
bbl=bar(dailymean7_urban.Timestamps,...
    round([dailymean7_urban.PM25 dailymean7_rural.PM25]),...
    1,'EdgeColor',[0 .9 .9],'LineWidth',1.5);
axis padded
tickMarks = {'XTick',[]};
set(gca,tickMarks{:});
xtips1 = bbl(1).XEndPoints;
ytips1 = bbl(1).YEndPoints;
labels1 = string(bbl(1).YData);
text(xtips1,ytips1,labels1,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 10,'rotation',45)
xtips2 = bbl(2).XEndPoints;
ytips2 = bbl(2).YEndPoints;
labels2 = string(bbl(2).YData);
text(xtips2,ytips2,labels2,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 8,'rotation',45)
title('7 day mean')
ylabel('PM25 (Î¼g/m^3)')
legend({'urban','rural'},'Location','southeast')
%-------------------------------------------------| PLOT - PM1
nexttile
bbl=bar(dailymedian_urban.Timestamps,...
    [dailymedian_urban.PM1 dailymedian_rural.PM1],...
    1,'EdgeColor',[0 .9 .9],'LineWidth',1.5);
axis padded
xtips1 = bbl(1).XEndPoints;
ytips1 = bbl(1).YEndPoints;
labels1 = string(bbl(1).YData);
text(xtips1,ytips1,labels1,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 10,'rotation',45)
xtips2 = bbl(2).XEndPoints;
ytips2 = bbl(2).YEndPoints;
labels2 = string(bbl(2).YData);
text(xtips2,ytips2,labels2,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 8,'rotation',45)
title('median')
ylabel('PM1 (Î¼g/m^3)')
legend({'urban','rural'},'Location','southeast')
%---
nexttile
bbl=bar(dailymax_urban.Timestamps,...
    [dailymax_urban.PM1 dailymax_rural.PM1],...
    1,'EdgeColor',[0 .9 .9],'LineWidth',1.5);
axis padded
xtips1 = bbl(1).XEndPoints;
ytips1 = bbl(1).YEndPoints;
labels1 = string(bbl(1).YData);
text(xtips1,ytips1,labels1,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 10,'rotation',45)
xtips2 = bbl(2).XEndPoints;
ytips2 = bbl(2).YEndPoints;
labels2 = string(bbl(2).YData);
text(xtips2,ytips2,labels2,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 8,'rotation',45)
title('max')
ylabel('PM1 (Î¼g/m^3)')
legend({'urban','rural'},'Location','southeast')
%---
nexttile
bbl=bar(dailymin_urban.Timestamps,...
    [dailymin_urban.PM1 dailymin_rural.PM1],...
    1,'EdgeColor',[0 .9 .9],'LineWidth',1.5);
axis padded
xtips1 = bbl(1).XEndPoints;
ytips1 = bbl(1).YEndPoints;
labels1 = string(bbl(1).YData);
text(xtips1,ytips1,labels1,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 10,'rotation',45)
xtips2 = bbl(2).XEndPoints;
```

```
ytips2 = bb1(2).YEndPoints;
labels2 = string(bb1(2).YData);
text(xtips2,ytips2,labels2,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 8,'rotation',45)
title('min')
ylabel('PM1 (Î¼g/m^3)')
legend({'urban','rural'},'Location','southeast')
%---
nexttile
bb1=bar(dailymean_urban.Timestamps,...
    round([dailymean_urban.PM1 dailymean_rural.PM1]),...
    1,'EdgeColor',[0 .9 .9],'LineWidth',1.5);
axis padded
xtips1 = bb1(1).XEndPoints;
ytips1 = bb1(1).YEndPoints;
labels1 = string(bb1(1).YData);
text(xtips1,ytips1,labels1,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 10,'rotation',45)
xtips2 = bb1(2).XEndPoints;
ytips2 = bb1(2).YEndPoints;
labels2 = string(bb1(2).YData);
text(xtips2,ytips2,labels2,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 8,'rotation',45)
title('mean')
ylabel('PM1 (Î¼g/m^3)')
legend({'urban','rural'},'Location','southeast')
%---
nexttile
bb1=bar(dailymean7_urban.Timestamps,...
    round([dailymean7_urban.PM1 dailymean7_rural.PM1]),...
    1,'EdgeColor',[0 .9 .9],'LineWidth',1.5);
axis padded
tickMarks = {'XTick',[]};
set(gca,tickMarks{:});
xtips1 = bb1(1).XEndPoints;
ytips1 = bb1(1).YEndPoints;
labels1 = string(bb1(1).YData);
text(xtips1,ytips1,labels1,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 10,'rotation',45)
xtips2 = bb1(2).XEndPoints;
ytips2 = bb1(2).YEndPoints;
labels2 = string(bb1(2).YData);
text(xtips2,ytips2,labels2,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 8,'rotation',45)
title('7 day mean')
ylabel('PM1 (Î¼g/m^3)')
legend({'urban','rural'},'Location','southeast')
```

ThingSpeak MATLAB code for Visualization of values of TVOC + eCO2 readings:

```
myid = XXXXX;
myfield = 1;
mykey = 'XXXXX';



%{
[mydata,mytimestamps] = thingSpeakRead(myid, ...
    'Fields',[1 2 3 4 5 6 7 8], ...
    'numDays',3, ...
    'ReadKey',mykey, ...
    OutputFormat='timetable')
%}



mydata_rural= thingSpeakRead(myid, ...
    'Fields',[1 2 3 4 5 6 7 8], ...
    'ReadKey',mykey, ...
    OutputFormat='timetable', ...
    DateRange=[datetime('2022-06-01'),datetime('2022-06-08')]);


mydata_urban= thingSpeakRead(myid, ...
    'Fields',[1 2 3 4 5 6 7 8], ...
    'ReadKey',mykey, ...
    OutputFormat='timetable', ...
    DateRange=[datetime('2022-06-20'),datetime('2022-06-27')]);


dailymax_rural = retime(mydata_rural, 'daily', 'max');
dailymin_rural = retime(mydata_rural, 'daily', 'min');
dailymean_rural = retime(mydata_rural, 'daily', 'mean');
dailymean7_rural = retime(mydata_rural, 'regular','mean', 'TimeStep',days(7));
dailymedian_rural = retime(mydata_rural, 'daily', 'median');


dailymax_urban = retime(mydata_urban, 'daily', 'max');
dailymin_urban = retime(mydata_urban, 'daily', 'min');
dailymean_urban = retime(mydata_urban, 'daily', 'mean');
dailymean7_urban = retime(mydata_urban, 'regular','mean', 'TimeStep',days(7));
dailymedian_urban = retime(mydata_urban, 'daily', 'median');



%------------------------------------------------| PLOT


tiledlayout(2,5)



%------------------------------------------------| PLOT - TVOC
nexttile
bb1=bar(dailymedian_urban.Timestamps,...
    [dailymedian_urban.TVOC dailymedian_rural.TVOC],...
    1,'EdgeColor',[0 .9 .9],'LineWidth',1.5);
axis padded
xtips1 = bb1(1).XEndPoints;
ytips1 = bb1(1).YEndPoints;
labels1 = string(bb1(1).YData);
text(xtips1,ytips1,labels1,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 10,'rotation',45)
xtips2 = bb1(2).XEndPoints;
ytips2 = bb1(2).YEndPoints;
```

```
labels2 = string(bb1(2).YData);
text(xtips2,ytips2,labels2,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 8,'rotation',45)


title('median')
ylabel('TVOC (ppb)')
legend({'urban','rural'},'Location','southeast')


%---


nexttile
bb1=bar(dailymax_urban.Timestamps,...
    [dailymax_urban.TVOC dailymax_rural.TVOC],...
    1,'EdgeColor',[0 .9 .9],'LineWidth',1.5);
axis padded
xtips1 = bb1(1).XEndPoints;
ytips1 = bb1(1).YEndPoints;
labels1 = string(bb1(1).YData);
text(xtips1,ytips1,labels1,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 10,'rotation',45)
xtips2 = bb1(2).XEndPoints;
ytips2 = bb1(2).YEndPoints;
labels2 = string(bb1(2).YData);
text(xtips2,ytips2,labels2,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 8,'rotation',45)


title('max')
ylabel('TVOC (ppb)')
legend({'urban','rural'},'Location','southeast')


%---


nexttile
bb1=bar(dailymin_urban.Timestamps,...
    [dailymin_urban.TVOC dailymin_rural.TVOC],...
    1,'EdgeColor',[0 .9 .9],'LineWidth',1.5);
axis padded
xtips1 = bb1(1).XEndPoints;
ytips1 = bb1(1).YEndPoints;
labels1 = string(bb1(1).YData);
text(xtips1,ytips1,labels1,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 10,'rotation',45)
xtips2 = bb1(2).XEndPoints;
ytips2 = bb1(2).YEndPoints;
labels2 = string(bb1(2).YData);
text(xtips2,ytips2,labels2,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 8,'rotation',45)


title('min')
ylabel('TVOC (ppb)')
legend({'urban','rural'},'Location','southeast')


%---


nexttile
bb1=bar(dailymean_urban.Timestamps,...
    round([dailymean_urban.TVOC dailymean_rural.TVOC]),...
    1,'EdgeColor',[0 .9 .9],'LineWidth',1.5);
axis padded
xtips1 = bb1(1).XEndPoints;
```

```
ytips1 = bb1(1).YEndPoints;
labels1 = string(bb1(1).YData);
text(xtips1,ytips1,labels1,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 10,'rotation',45)
xtips2 = bb1(2).XEndPoints;
ytips2 = bb1(2).YEndPoints;
labels2 = string(bb1(2).YData);
text(xtips2,ytips2,labels2,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 8,'rotation',45)


title('mean')
ylabel('TVOC (ppb)')
legend({'urban','rural'},'Location','southeast')


%---


nexttile
bb1=bar(dailymean7_urban.Timestamps,...
    round([dailymean7_urban.TVOC dailymean7_rural.TVOC]),...
    1,'EdgeColor',[0 .9 .9],'LineWidth',1.5);
axis padded
tickMarks = {'XTick',[]};
set(gca,tickMarks{:});
xtips1 = bb1(1).XEndPoints;
ytips1 = bb1(1).YEndPoints;
labels1 = string(bb1(1).YData);
text(xtips1,ytips1,labels1,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 10,'rotation',45)
xtips2 = bb1(2).XEndPoints;
ytips2 = bb1(2).YEndPoints;
labels2 = string(bb1(2).YData);
text(xtips2,ytips2,labels2,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 8,'rotation',45)


title('7 day mean')
ylabel('TVOC (ppb)')
legend({'urban','rural'},'Location','southeast')


%-------------------------------------------------| PLOT - eCO2
nexttile
bb1=bar(dailymedian_urban.Timestamps,...
    [dailymedian_urban.eCO2 dailymedian_rural.eCO2],...
    1,'EdgeColor',[0 .9 .9],'LineWidth',1.5);
axis padded
xtips1 = bb1(1).XEndPoints;
ytips1 = bb1(1).YEndPoints;
labels1 = string(bb1(1).YData);
text(xtips1,ytips1,labels1,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 10,'rotation',45)
xtips2 = bb1(2).XEndPoints;
ytips2 = bb1(2).YEndPoints;
labels2 = string(bb1(2).YData);
text(xtips2,ytips2,labels2,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 8,'rotation',45)


title('median')
ylabel('eCO2 (ppm)')
legend({'urban','rural'},'Location','southeast')


%---
```

```
nexttile
bb1=bar(dailymax_urban.Timestamps,...
    [dailymax_urban.eCO2 dailymax_rural.eCO2],...
    1,'EdgeColor',[0 .9 .9],'LineWidth',1.5);
axis padded
xtips1 = bb1(1).XEndPoints;
ytips1 = bb1(1).YEndPoints;
labels1 = string(bb1(1).YData);
text(xtips1,ytips1,labels1,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 10,'rotation',45)
xtips2 = bb1(2).XEndPoints;
ytips2 = bb1(2).YEndPoints;
labels2 = string(bb1(2).YData);
text(xtips2,ytips2,labels2,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 8,'rotation',45)


title('max')
ylabel('eCO2 (ppm)')
legend({'urban','rural'},'Location','southeast')


%---


nexttile
bb1=bar(dailymin_urban.Timestamps,...
    [dailymin_urban.eCO2 dailymin_rural.eCO2],...
    1,'EdgeColor',[0 .9 .9],'LineWidth',1.5);
axis padded
xtips1 = bb1(1).XEndPoints;
ytips1 = bb1(1).YEndPoints;
labels1 = string(bb1(1).YData);
text(xtips1,ytips1,labels1,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 10,'rotation',45)
xtips2 = bb1(2).XEndPoints;
ytips2 = bb1(2).YEndPoints;
labels2 = string(bb1(2).YData);
text(xtips2,ytips2,labels2,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 8,'rotation',45)


title('min')
ylabel('eCO2 (ppm)')
legend({'urban','rural'},'Location','southeast')


%---


nexttile
bb1=bar(dailymean_urban.Timestamps,...
    round([dailymean_urban.eCO2 dailymean_rural.eCO2]),...
    1,'EdgeColor',[0 .9 .9],'LineWidth',1.5);
axis padded
xtips1 = bb1(1).XEndPoints;
ytips1 = bb1(1).YEndPoints;
labels1 = string(bb1(1).YData);
text(xtips1,ytips1,labels1,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 10,'rotation',45)
xtips2 = bb1(2).XEndPoints;
ytips2 = bb1(2).YEndPoints;
labels2 = string(bb1(2).YData);
text(xtips2,ytips2,labels2,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 8,'rotation',45)


title('mean')
ylabel('eCO2 (ppm)')
```

```
legend({'urban','rural'},'Location','southeast')


%---


nexttile
bb1=bar(dailymean7_urban.Timestamps,...
    round([dailymean7_urban.eCO2 dailymean7_rural.eCO2]),...
    1,'EdgeColor',[0 .9 .9],'LineWidth',1.5);
axis padded
tickMarks = {'XTick',[]};
set(gca,tickMarks{:});
xtips1 = bb1(1).XEndPoints;
ytips1 = bb1(1).YEndPoints;
labels1 = string(bb1(1).YData);
text(xtips1,ytips1,labels1,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 10,'rotation',45)
xtips2 = bb1(2).XEndPoints;
ytips2 = bb1(2).YEndPoints;
labels2 = string(bb1(2).YData);
text(xtips2,ytips2,labels2,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 8,'rotation',45)


title('7 day mean')
ylabel('eCO2 (ppm)')
legend({'urban','rural'},'Location','southeast')
```

ThingSpeak MATLAB code for Visualization of values of Sound Level readings:

```
myid = XXXXX;
myfield = 1;
mykey = 'XXXXX';


%{
[mydata,mytimestamps] = thingSpeakRead(myid, ...
    'Fields',[1 2 3 4 5 6 7 8], ...
    'numDays',3, ...
    'ReadKey',mykey, ...
    OutputFormat='timetable')
%}


mydata_rural= thingSpeakRead(myid, ...
    'Fields',[1 2 3 4 5 6 7 8], ...
    'ReadKey',mykey, ...
    OutputFormat='timetable', ...
    DateRange=[datetime('2022-06-01'),datetime('2022-06-08')]);

mydata_urban= thingSpeakRead(myid, ...
    'Fields',[1 2 3 4 5 6 7 8], ...
    'ReadKey',mykey, ...
    OutputFormat='timetable', ...
    DateRange=[datetime('2022-06-20'),datetime('2022-06-27')]);

dailymax_rural = retime(mydata_rural, 'daily', 'max');
dailymin_rural = retime(mydata_rural, 'daily', 'min');
dailymean_rural = retime(mydata_rural, 'daily', 'mean');
dailymean7_rural = retime(mydata_rural, 'regular','mean', 'TimeStep',days(7));
dailymedian_rural = retime(mydata_rural, 'daily', 'median');

dailymax_urban = retime(mydata_urban, 'daily', 'max');
dailymin_urban = retime(mydata_urban, 'daily', 'min');
dailymean_urban = retime(mydata_urban, 'daily', 'mean');
dailymean7_urban = retime(mydata_urban, 'regular','mean', 'TimeStep',days(7));
dailymedian_urban = retime(mydata_urban, 'daily', 'median');


%------------------------------------------------| PLOT

tiledlayout(1,5)

%------------------------------------------------| PLOT - SoundLeveldB
nexttile
bb1=bar(dailymedian_urban.Timestamps,...
    [dailymedian_urban.SoundLeveldB dailymedian_rural.SoundLeveldB],...
    1,'EdgeColor',[0 .9 .9],'LineWidth',1.5);
axis padded
xtips1 = bb1(1).XEndPoints;
ytips1 = bb1(1).YEndPoints;
labels1 = string(bb1(1).YData);
text(xtips1,ytips1,labels1,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 10,'rotation',45)
xtips2 = bb1(2).XEndPoints;
ytips2 = bb1(2).YEndPoints;
labels2 = string(bb1(2).YData);
text(xtips2,ytips2,labels2,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 8,'rotation',45)

title('median')
ylabel('SoundLeveldB (dB)')
legend({'urban','rural'},'Location','southeast')

%---

nexttile
bb1=bar(dailymax_urban.Timestamps,...
    [dailymax_urban.SoundLeveldB dailymax_rural.SoundLeveldB],...
```

```
    1,'EdgeColor',[0 .9 .9],'LineWidth',1.5);
axis padded
xtips1 = bbl(1).XEndPoints;
ytips1 = bbl(1).YEndPoints;
labels1 = string(bbl(1).YData);
text(xtips1,ytips1,labels1,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 10,'rotation',45)
xtips2 = bbl(2).XEndPoints;
ytips2 = bbl(2).YEndPoints;
labels2 = string(bbl(2).YData);
text(xtips2,ytips2,labels2,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 8,'rotation',45)

title('max')
ylabel('SoundLeveldB (dB)')
legend({'urban','rural'},'Location','southeast')

%---

nexttile
bbl=bar(dailymin_urban.Timestamps,...
    [dailymin_urban.SoundLeveldB dailymin_rural.SoundLeveldB],...
    1,'EdgeColor',[0 .9 .9],'LineWidth',1.5);
axis padded
xtips1 = bbl(1).XEndPoints;
ytips1 = bbl(1).YEndPoints;
labels1 = string(bbl(1).YData);
text(xtips1,ytips1,labels1,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 10,'rotation',45)
xtips2 = bbl(2).XEndPoints;
ytips2 = bbl(2).YEndPoints;
labels2 = string(bbl(2).YData);
text(xtips2,ytips2,labels2,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 8,'rotation',45)

title('min')
ylabel('SoundLeveldB (dB)')
legend({'urban','rural'},'Location','southeast')

%---

nexttile
bbl=bar(dailymean_urban.Timestamps,...
    round([dailymean_urban.SoundLeveldB dailymean_rural.SoundLeveldB]),...
    1,'EdgeColor',[0 .9 .9],'LineWidth',1.5);
axis padded
xtips1 = bbl(1).XEndPoints;
ytips1 = bbl(1).YEndPoints;
labels1 = string(bbl(1).YData);
text(xtips1,ytips1,labels1,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 10,'rotation',45)
xtips2 = bbl(2).XEndPoints;
ytips2 = bbl(2).YEndPoints;
labels2 = string(bbl(2).YData);
text(xtips2,ytips2,labels2,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 8,'rotation',45)

title('mean')
ylabel('SoundLeveldB (dB)')
legend({'urban','rural'},'Location','southeast')

%---

nexttile
bbl=bar(dailymean7_urban.Timestamps,...
    round([dailymean7_urban.SoundLeveldB dailymean7_rural.SoundLeveldB]),...
    1,'EdgeColor',[0 .9 .9],'LineWidth',1.5);
axis padded
tickMarks = {'XTick',[]};
```

```
set(gca,tickMarks{:});
xtips1 = bbl(1).XEndPoints;
ytips1 = bbl(1).YEndPoints;
labels1 = string(bbl(1).YData);
text(xtips1,ytips1,labels1,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 10,'rotation',45)
xtips2 = bbl(2).XEndPoints;
ytips2 = bbl(2).YEndPoints;
labels2 = string(bbl(2).YData);
text(xtips2,ytips2,labels2,'HorizontalAlignment','left',...
    'VerticalAlignment','bottom','FontSize', 8,'rotation',45)

title('7 day mean')
ylabel('SoundLeveldB (dB)')
legend({'urban','rural'},'Location','southeast')
```