

POLITECNICO DI TORINO

Master's degree course in Electronic Engineering

Master's Degree Thesis

**Studying and developing
a monitoring system for urban sustainability**



Supervisor

Prof. Maurizio Rebaudengo

Prof. Bartolomeo Montrucchio

Candidate

Mingyang Zhang

September 2018

Acknowledgment

I want to dedicate this work to my wife, for supporting me all the time.

Thanks to Prof. Maurizio Rebaudengo and Prof. Bartolomeo Montrucchio, who gave me this opportunity and taught me a lot of things. I have been able to deepen my knowledge of sensors, Linux programming, data visualization in this project.

Thanks to Edoardo Giusto and Mohammad Vakili, who helped me a lot during this thesis. It was a fantastic experience and I have grown up a lot and learned a lot working with both of you during last months.

Contents

Introduction	4
Chapter 1 IoT	6
1.1 Wireless Sensor Networks	8
1.1.1 Applications of WSN	9
1.1 Operating Systems used in IoT	10
1.1.2 TinyOS	12
1.1.3 Contiki.....	13
1.1.4 μ C/OS-II.....	14
1.1.5 Linux on embedded systems	15
Chapter 2 Air Pollution Monitoring.....	18
2.1 Air pollution.....	18
2.1.1 Air Pollution in China	20
2.1.2 Air Pollution in Europe	21
2.2 Available Air Pollution Monitoring Platforms.....	22
2.1.3 Libelium	22
2.1.4 uRADMonitor	25
Chapter 3 Design and Development	27
3.1 Design of System Architecture	27

3.2	Hardware.....	29
3.2.1	Sensors	30
3.2.2	Boards.....	34
3.2.3	Casing.....	38
3.3	Software	39
3.3.1	Raspberry Pi side.....	41
3.3.2	Server side	47
3.3.3	Visualization interface.....	52
Chapter 4	Experimental Data Analysis	55
4.1	Correlation Analysis of PM Sensors	55
4.2	Test of other sensors.....	58
Chapter 5	Future Work.....	60
Chapter 6	Conclusion.....	62
	Bibliography	64

Introduction

With the development of industrialization and urbanization in contemporary society, the air pollution is getting more and more serious. So there is an urgent need to develop a set of scalable, easy-to-maintain, safe and secure air pollution monitoring system that can be in real time. Currently, there are two main methods for monitoring urban air pollution:

- 1) Traditional manual sampling with the analytical methods used in laboratories;
- 2) Automated air environment monitoring systems using equipment with online monitoring abilities.

The first method takes a long time to sample and is susceptible to human activities. Moreover, a high concentration of harmful gases may harm the workers at the scene.

This article uses the wireless sensor networks(WSN) for air pollution monitoring. The monitoring system can well solve the problems of the above methods. Wireless sensor network has the following advantages in the field of air pollution monitoring:

- 1) Simple deployment, flexible, monitoring nodes are mobile;
- 2) Robust;
- 3) Easy to operate and maintain, new sensor nodes can be automatically added to the monitoring system.

This thesis proposes an air pollution monitoring system based on WSN. Moreover, it explains in detail the system architecture, software and hardware design.

Chapter 1 IoT

"Internet of Things" refers to a huge network formed by the combination of internet and various information sensing devices, such as radio frequency identification (RFID) devices, infrared sensors, global positioning systems, laser scanners. Its purpose is to make all items connected to the network. The systems can be automatically, in real-time identify, locate, track, monitor, and trigger events of things. "Internet of Things" is another revolution of the world information industry followed by computers, the Internet and mobile communication networks.

The concept of "Internet of Things" broke the traditional idea which has always been the separation of physical infrastructures and IT infrastructures: on the one hand, there are airports, roads and buildings, on the other hand, there are data centers and personal computers. In the "Internet of Things" era, reinforced concrete, cables will be integrated into a unified infrastructure with chips and broadband.

The concept of "Internet of Things" was introduced in 1999. Its definition was straightforward: connect all items with the Internet by using information sensing devices such as RF identification for intelligent identification and management. That is to say that the Internet of Things is a new technology that connects various sensors and the existing Internet.

If we had computers that knew everything there was to know about things—using data they gathered without any help from us—we would be able to track and count everything, and significantly reduce waste, loss, and cost.¹ In 2005, the International Telecommunication Union (ITU) released the "ITU Internet Report 2005: Internet of Things". The report points out that the ubiquitous Internet communication era is coming. Information about all objects in the world, from tires, toothbrushes, homes to paper towels, can be actively transmitted through the Internet. Radio Frequency Identification (RFID), sensor technology, nanotechnology, intelligent embedded technology will be used in a wider range of applications.

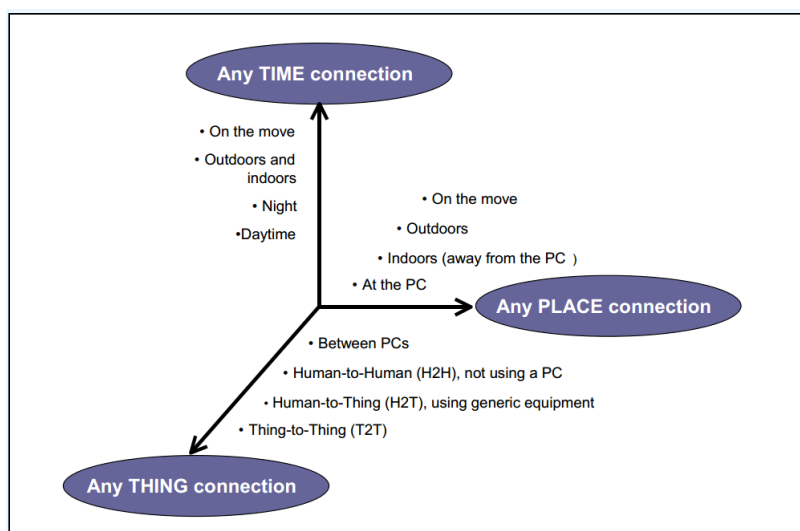


Figure 1.1: Introducing a new dimension to the telecommunication environment²

The world's first international IoT conference was held in Zurich in March 2008. It was called "Internet of Things 2008" explores how the new concept of "Internet of Things" and new technologies can advance the "Internet of Things" to the next stage of development. After Barack Obama became the US president, he held a "Round Table" meeting with American business leaders. As one of the only two representatives, the CEO of IBM, Samuel Palmisano first proposed the concept of "smart planet." He suggested the new government should invest in a new generation of smart infrastructure. He also clarified its short-term and long-term benefits. Barack Obama gave a positive response to this: "The stimulus funds will be invested in emerging technologies such as broadband networks. There is no doubt that this is the opportunity of the United States to maintain and regain a competitive advantage in the 21st century." Once this concept was put forward, it attracted high attention in the United States. Moreover, even some analysts believe that IBM's concept may become the US national strategy and cause a worldwide sensation.

1.1 Wireless Sensor Networks

The IoT presents itself practically as a set of Wireless Sensor Networks (WSNs).

A wireless sensor network (WSN) is a network formed by a large number of sensor nodes where each node is equipped with a sensor to detect physical phenomena such as light, heat, pressure, etc.³ It is related to the development of the economic development and the security because of its characteristics of flexible networking. The Internet of

Things is through a variety of sensors on the object and the wireless network of them, in the end, perceives the entire material world.

The essential components of a sensor network node include the following basic units: sensing unit (sensor(s) and analog to digital converter(s)), processing unit (CPU, memory, embedded operating system), the communication unit (wireless communication modules) and power supply. Also, other functional units that can also be included: a positioning system, a mobile system, and a self-powered power supply system, etc. In a sensor network, nodes can be deployed in large quantities using aircraft deployment or manual laying. It is planted inside or near the object to be perceived. These nodes form a wireless network by self-organizing, and perceive, collect, and process the information in the network coverage area in a collaborative way in real-time. Then the data is transferred to the remote control management center. On the other hand, the remote management center can also control and manipulate the network nodes in real time.

1.1.1 Applications of WSN

With the IoT industry growing faster and faster, research of WSN is quite popular in universities and institutions. As it is shown in Figure 1.1, the number of installed sensors is growing rapidly. So the WSN research is quite commercially valuable with this growing market. Currently, WSN technology research for the Internet of Things includes the following aspects:

- 1) Advanced testing technology and networked measurement and control;
- 2) Research on intelligent sensor network node;

- 3) Research on sensor network organization structure and underlying protocol;
- 4) Detection and control of the sensor network itself;
- 5) Security of the sensor network.

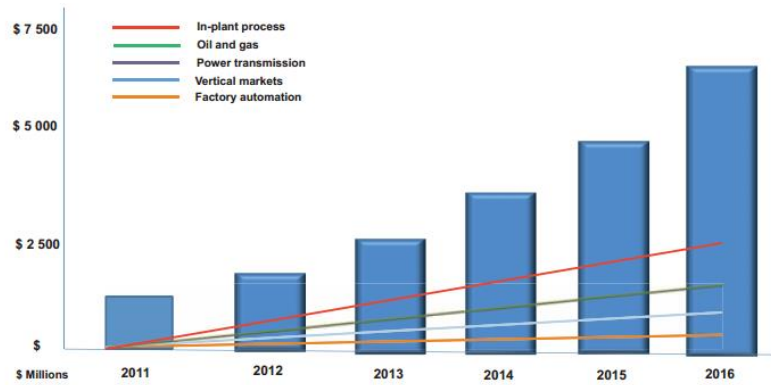


Figure 1.2: Global installed industrial wireless sensing points³

1.2 Operating Systems used in IoT

For good portability, ease of development and maintenance, an operating system is needed for the WSN. An operating system not only can help developers control the hardware, manipulate with interrupts and scheduled tasks, but also provides universal functions like APIs, GUIs, network protocols and so on.

For a WSN, the operating system should have the following characteristics:

- Lite, low-power. This is because the power energy, communication capability and the computing capacity of the sensor node are limited. The sensor is battery powered and has limited energy, so energy efficient design is critical. Wireless sensor network usually transmits data in a "multi-hop" manner with a communication range of only a few tens of meters. A sensor

node has a limit of memory, computing capabilities due to size, cost, and energy. These constraints require the operating system not only to be small, have the ability to run under limited resources, but also to save energy when dealing with data processing and data communication.

- Reconfigurable, robust, and fault-tolerant. This is because the number of sensor nodes can reach several million and the network often has new nodes joining or existing nodes failing. This feature of WSN requires the sensor node operating system to have reconfigurability and adaptability, high robustness and fault tolerance. When the network topology changes, the operating system must be able to respond to this change and actively update itself when needed.
- Good portability. Different sensor network applications are concerned with different physical quantities, so the requirements for the electronic system can be quite different. Hence, the development of the hardware platform, software system, and network protocol can be very different. This difficulty requires the operating system to have good portability, to work under a variety of hardware platforms, and to provide a variety of different functions to meet actual needs.
- Modular. The trend of wireless sensor network design is not to make a unique platform for a specific application. With the widespread use of wireless sensor networks, the changes in node composition are enormous. On a

specific hardware platform, it is critical to quickly and conveniently combine software modules to implement applications according to different scenarios.

1.2.1 TinyOS

TinyOS is an open source micro operating system developed by UC Berkeley. It is designed for wireless sensor networks and currently dominates the wireless sensor network operating system field. Its component-based architecture enables it to implement a variety of applications quickly. Moreover, its component libraries include network protocols, distributed services, sensor drivers, and data acquisition tools. A complete application system is made up of these libraries. Unused components will not be introduced, thus reducing the memory requirements. TinyOS uses an event-driven model that handles high-concurrency events in a small space and saves energy because the CPU does not need to look for events of interest actively.

At present, TinyOS can run on many hardware platforms. The hardware platforms that exposes schematics on the TinyOS website are Telos (Rev A) and Telos (Rev B), Mica2 Dot, Mica2, Mica. There are also some commercial and non-commercial organizations which also have some hardware platforms to run TinyOS. TinyOS is also widely used in practical projects, and dozens of projects using TinyOS are listed on its official website and are continually being updated.

The researchers from UC Berkeley present four broad requirements which motivate the design of TinyOS:⁴

- Limited resources: due to the goals of small size, low cost, and low power consumption of sensor nodes.
- Reactive Concurrency: concurrent tasks are needed to be dealt with on the sensor node, it requires an approach to concurrency management that reduces potential bugs while respecting resource and timing constraints.
- Flexibility: The variation in hardware and applications and the rate of innovation requires a flexible OS that is both application-specific to reduce space and power, and independent of the boundary between hardware and software.
- Low Power: Demands of size and cost, as well as untethered operation make low-power operation a key goal of mote design.

1.2.2 Contiki

Contiki is a multi-tasking operating system developed by Adam Dunkels from the Swedish Computer Science Research Institute, for embedded systems such as network sensor nodes with minimal memory resources. Contiki is written entirely in C language, and the source code is open. It supports network interconnection, is highly portable, and has a tiny amount of code. It supports embedded systems from 8-bit microcontrollers to old-fashioned 8-bit home computers. Since its launch in May 2003, Contiki has been ported to 20 hardware platforms of different types.

Contiki provides a simple event-driven kernel that supports prototyping processes and optional preemptive multitasking. It delivers messages for inter-task communication,

has a dynamic process structure, and supports loading and unloading programs. Using uIP to implement the native TCP/IP protocol, one can implement a graphical interface system on directly connected terminals and terminals connected through a network. At present, a basic system of Contiki (supporting multitasking, network and graphical interface) has a compiled code size of 32K, and a system that fully supports the Web server, Web browser, and other functions have a compiled code size of about 64K. The smallest system currently running Contiki has only 2K RAM, which runs the basic system, web server, virtual network computer server, and a small virtual desktop.

1.2.3 μ C/OS-II

The μ C/OS-II operating system is a free embedded operating system with excellent performance, open source code, and wide application. In July 2002, μ C/OS-II received the US Federal Aviation Administration's certification for commercial aircraft in compliance with the RTCA DO-178B standard. It is a compact, real-time operating system with a deprived real-time kernel. The kernel provides tasks such as task scheduling and management, time management, synchronization and communication between tasks, memory management, and interrupt services. It is portable, scalable, configurable, and stable.

μ C/OS-II allows defining several functions in C, each of which can execute as an independent thread or task. Each task runs at a different priority and runs as if it owns the CPU. Higher priority tasks can preempt lower priority tasks at any time. Higher priority tasks use the operating system (OS) services (such as a delay or event) to allow lower

priority tasks to execute. OS services are provided for managing tasks and memory, communicating between tasks, and timing. ⁵

1.2.4 Linux on embedded systems

It is a group of different operating systems based on the Linux kernel. They are used in embedded systems. As a real operating system (by the traditional definition), the most significant advantage of using embedded Linux is that its source code is open and follows the GPL.

Also, embedded Linux inherits unlimited open source resources on the Internet and has the features of embedded operating systems at the same time. Embedded Linux has the features of free copyright fees, excellent performance, easy software porting, open code, many application software support, product development cycle short. New products can come on the market quickly because many open codes can be referenced and ported.

As it is easy to develop and there is much code to take as a reference, we chose Linux as the operating system. Particularly in this project, we use the Arch Linux ARM, a lightweight and flexible Linux distribution for ARM computers. It has the following principles:⁶

- **Simplicity:** It ships software as released by the original developers (upstream) with minimal distribution-specific (downstream) changes.
- **Modernity:** Arch Linux strives to maintain the latest stable release versions of its software as long as systemic package breakage can be reasonably avoided.

- Pragmatism: Evidence-based technical analysis and debate are what matter, not politics or popular opinion.
- User centrality: The Arch Linux is targeted at the proficient GNU/Linux user, or anyone with a do-it-yourself attitude who is willing to read the documentation, and solve their problems.

Chapter 2 Air Pollution Monitoring

2.1 Air pollution

The so-called air pollution refers to the human or the nature discharging various pollutants into the atmosphere. Its content exceeds the affordability of the environment, which deteriorates the air quality and influences people's work, life, health, property, and ecological environment.

Sources of pollution can be classified into natural sources and human-made sources. Natural pollution sources refer to places or areas where natural pollutants are emitted into the atmosphere, such as active volcanoes that emit dust, sulfur dioxide, and hydrogen sulfide. Human-made pollution sources can be classified according to different methods. According to the spatial distribution of pollution sources, they can be divided into point pollution sources, surface pollution sources and regional pollution sources. According to people's social activities, they can be divided into domestic pollution sources, industrial pollution sources, traffic pollution sources and so on.

According to the form of existence, they can be divided into fixed pollution sources and mobile pollution sources.

Particulates are a source of air pollution which is highly concerned nowadays. They are microscopic solid or liquid matter suspended in the atmosphere of Earth.⁷

Researches in recent decades in the field of international environmental epidemiology have confirmed that long-term or short-term exposure to inhalable particulate matter (PM₁₀), especially PM_{2.5}, can lead to the increase of incidence of cardiopulmonary disease and mortality. The study of the relationship between national air pollution and mortality and morbidity in the United States shows that population mortality is related to the concentration of particulate matter before death.⁸

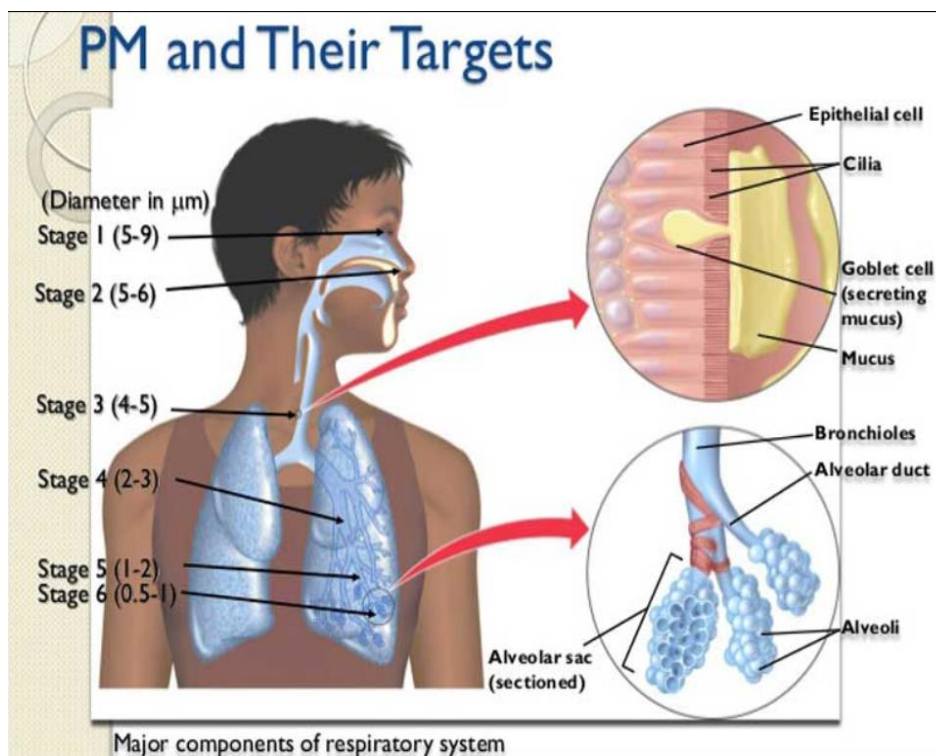


Figure 2.1: The penetration of particulate in the respiratory system⁹

2.1.1 Air Pollution in China

Currently, China is the country with the most severe air pollution in the world. The fall of Beijing had initially been a good season with white clouds and blue sky. However, for many years, the sky in Beijing has always been grey and even smoggy. It is difficult to see the blue sky and white clouds in the daytime, and the stars are not seen at night. China has been struggling to rein in air pollution ever since the late 1990s after Beijing won the bid for the 2008 Olympics.¹⁰ At the beginning of winter in 2011, the number of smoggy days in Beijing was even higher than average years. The air quality monitoring report issued by the environmental monitoring department is very different from the experience of the public. The PM_{2.5} monitored by the US Embassy in Beijing indicates that the air pollution situation in Beijing has repeatedly reached the "serious health hazard level" and the "dangerous level." As a result, "PM_{2.5}" quickly entered the public view and became a topic of widespread concern and discussion. Figure 2.2 shows an extremely PM_{2.5} polluted day in northeastern China.

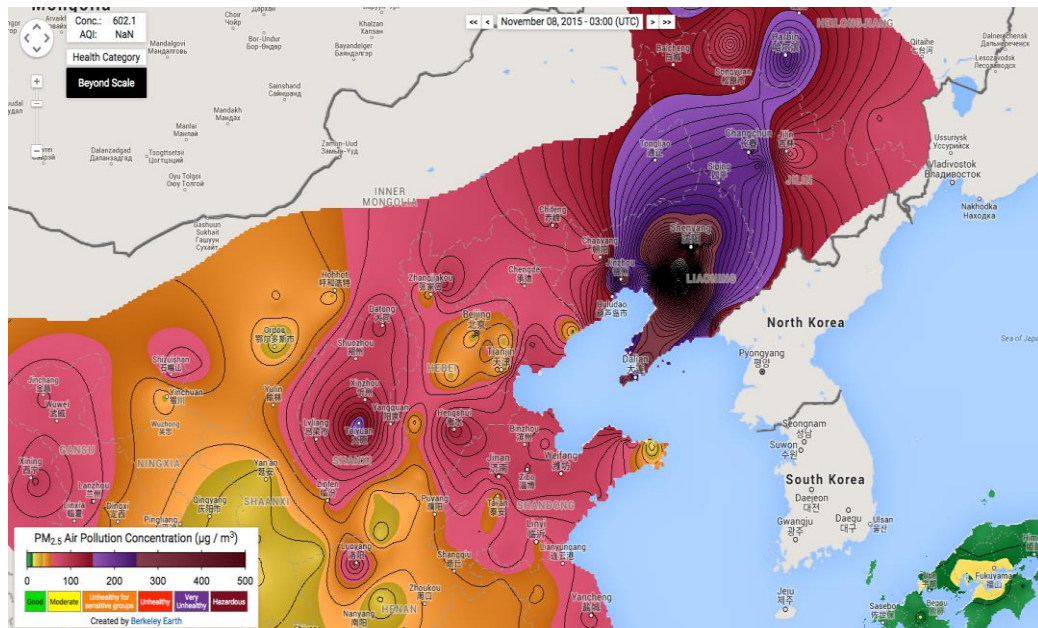


Figure 2.2: AQI in northeastern China on Sunday, November 8, 2015¹¹

2.1.2 Air Pollution in Europe

The situation is better in Europe. In Europe, emissions of many air pollutants have decreased substantially over the past decades, resulting in improved air quality across the region. However, air pollutant concentrations are still too high, and air quality problems persist.¹² European Union has a high standard of air quality for its member countries. Even with this high standard, Italy is one of the countries with the most polluted air in Europe, as it is shown in Figure 2.3. Moreover, Italy is the country in the European Union with the record number of premature deaths concerning the normal life expectancy due to air pollution. The estimate refers to a report of the European Environment Agency (EEA): in 2012 Italy recorded 84,400 deaths of this type, out of a total of 491,000 at an EU level.

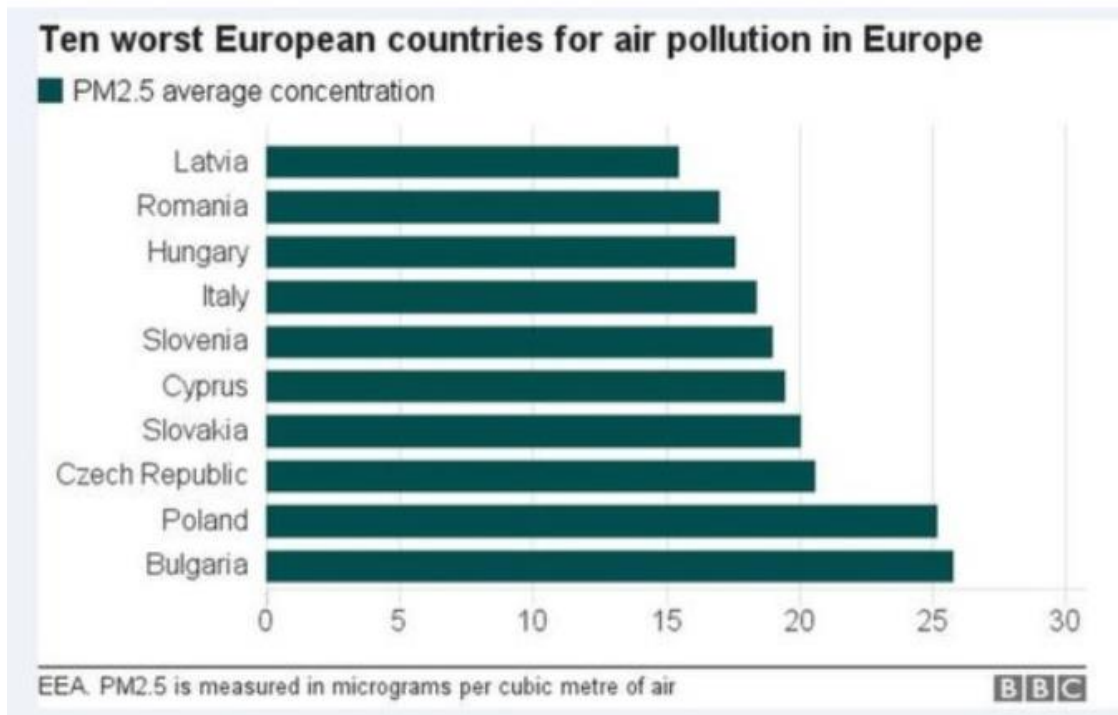


Figure 2.3: ten worst European countries for PM2.5 pollution¹³

2.2 Available Air Pollution Monitoring Platforms

2.1.3 Libelium

Libelium Waspnote is a leading commercial platform in this area. It is not only an air quality monitoring platform but a multifunctional platform. By using the "gases pro" module board, the developers can build an air pollution detecting system easily.



Figure 2.4: Libelium Wasp mote¹⁴

Main characteristics of Libelium Wasp mote:

- Robust waterproof IP65 enclosure
- Add or change a sensor probe in seconds
- Solar powered external panel option
- Radios available: 802.15.4, 868 MHz, 900 MHz, Wi-Fi, 4G, Sigfox and LoRaWAN
- Over the air programming (OTAP) of multiple nodes at once (via Wi-Fi or 4G radios)
- Special holders and brackets ready for installation in street lights and building fronts
- Graphical and intuitive interface Programming Cloud Service
- Built-in, 3-axes accelerometer

- External, contactless reset with a magnet
- Optional industrial protocols: RS-232, RS-485, Modbus, CAN Bus
- Optional GPS receiver
- Optional External Battery Module
- External SIM connector for the 4G models
- Fully certified: CE (Europe), FCC (USA), IC (Canada), ANATEL (Brazil), RCM (Australia), PTCRB (USA, cellular connectivity), AT&T (USA, cellular connectivity)



Figure 2.5: Libelium gases pro system¹⁵

2.1.4 uRADMonitor

uRADMonitor is another crucial player in the market of the air quality monitoring station.

It can detect eight different gas and several radiations. In general, this project is similar to the aim of our project. The manufacturer offers people several different models which aim at different detecting parameters and also use different communication between stations and the backend server. Therefore, customers can choose the corresponding model(s) for their applications.

Besides, they also provide a data management system which allows the customers to check all the collected data on a dashboard. This dashboard displays the parameters which the users want to check on a map with colors indicating number level for that gas/radiation.



Figure 2.6: uRADMonitor Modules with different internet connections

uRADMonitor has the ability to detect the following pollution:

- Temperature
- Pressure
- Humidity

- VOC
- Noise
- Formaldehyde
- Carbon Dioxide
- PM2.5
- PM10
- Radiation

Chapter 3 Design and Development

3.1 Design of System Architecture

This project aims to build a system which is lost-cost, low-power, real-time, with good visualization interface by using a single board computer. Main substances to be sampled are:

- Particulate matter 10 micrometres or less in diameter(PM10);
- Particulate matter 2.5 micrometres or less in diameter(PM2.5);
- Temperature;
- Atmospheric pressure;
- Relative humidity.

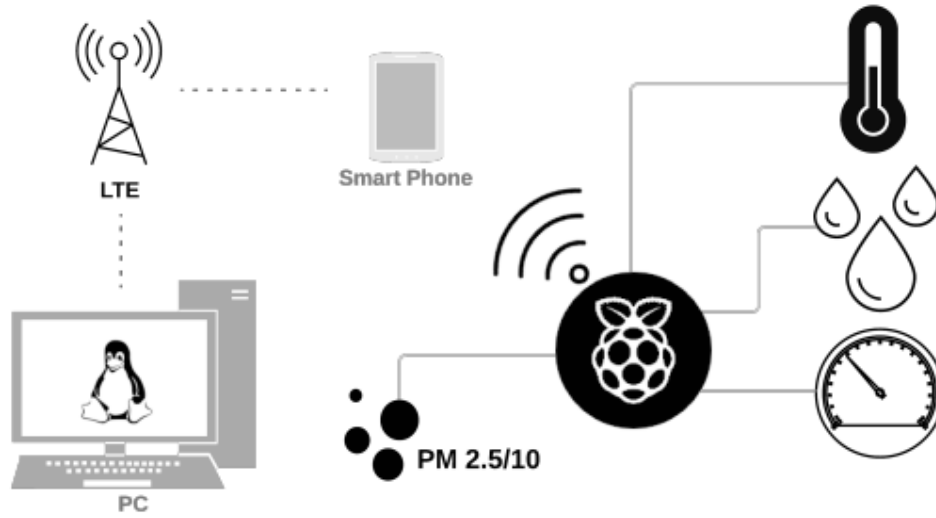


Figure 3.1: System architecture

The architecture of this system in Figure 3.2 is pretty simple. There is one single board computer (Raspberry Pi Zero W in this project) used as a medium between the sensor and the internet. This single board computer is connected to each sensor and samples data from them. Afterwards, the sampled data are sent to a remote server. The internet connection needed to execute the forwarding is provided by a hotspot network made by a smartphone in this case. Then the remote server will run an ad-hoc written software to parse the data and put them in a database designed for storing the data related all sensors. After that, a dashboard and graph composer, which is Grafana in this case¹⁶, will read data from the database and realize data visualization by making graphs, maps, etc.

As we want to make a mobile platform for monitoring the air quality, some fundamental requirements of this platform should have the following characteristics:

- **Low-cost:** it is crucial to choose components with a low budget in this very competitive industrial IoT market;
- **Low-power:** since this platform will be battery-powered, battery life is crucial;
- **Portable:** This means the size of the final product should be small. Otherwise, the mobile ability would be reduced;
- **Networkable:** We can only monitor the data in real time if the internet is available for the platforms;
- **Sensor calibration:** to ensure a reliable report of the atmospheric situation.

3.2 Hardware

Different sensors and boards were chosen to meet the design requirements of the platform. The final requirements of the hardware are:

- **Low-cost;**
- **Precise and stable,** to ensure a certified measurement;
- **WIFI / cellular data connection;**
- **Linux kernel based system,** ensuring high flexibility at the prototype stage;
- **Low energy consumption.**

The hardware part is divided into sensors and boards. For the particulate matter sensor, we chose HPMA115S0-XXX from Honeywell because it is laser-based (more precise than infrared ones) and easy to use (with an embedded MCU). We chose DHT22 for temperature and relative humidity detection because it is low-cost, small, and low-power.

3.2.1 Sensors

3.2.1.1 HPMA115S0-XXX - Particle Sensor

The Honeywell HPM Series Particle sensor is a laser-based sensing element that uses the light scattering technique to detect and count particles within the concentration range of 0 $\mu\text{g}/\text{m}^3$ to 1,000 $\mu\text{g}/\text{m}^3$ in a very given atmosphere. A laser light source illuminates a particle as it is pulled through the detection chamber. As particles undergo the ray, the light source becomes obscured and is recorded on a light detector. The light is then analyzed and converted to an electrical signal to calculate concentrations in real time. The Honeywell particle sensing element provides data on the particle concentration for a given particle concentration range.



Figure 3.3: HPMA115S0-XXX Particle Sensor

In this sensor, there is a fan to make the particles to be testes pass through the system and an air inlet with a fence to filter out undesirable particles.

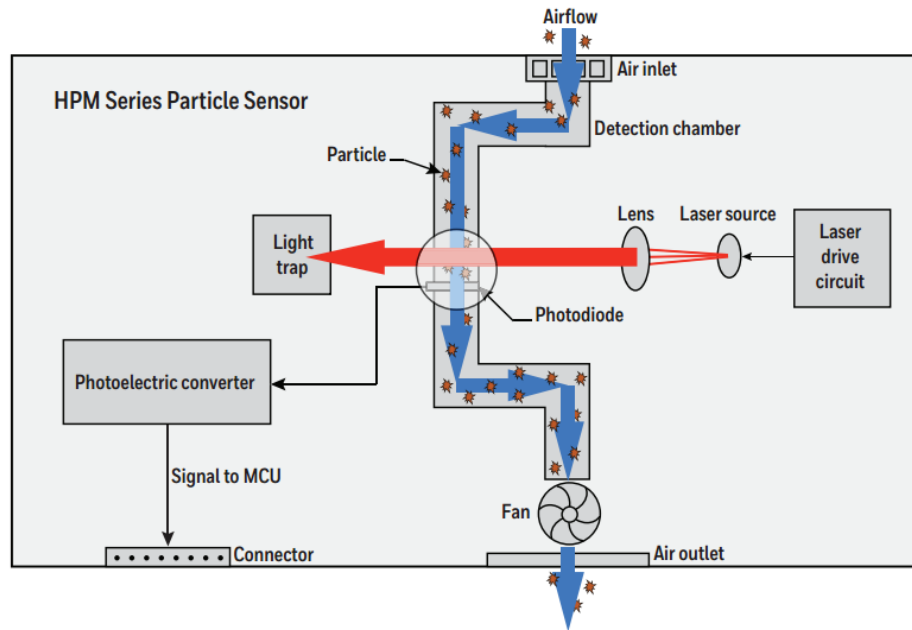


Figure 3.4: the internal structure of the particle sensor

Moreover, this sensor has an internal microcontroller, which uses UART (Universal Asynchronous Receiver Transmitter) to transfer data to external devices. The writing frequency on the serial port is about 1 Hz.

3.2.1.2 AM2302(DHT22) -Temperature and Relative Humidity Sensor

AM2302(DHT22) was chosen because it is a low-cost sensor with fairly precise measurement results. It applies exclusive digital-signal-collecting-technique and humidity sensing technology, assuring its reliability and stability. Its sensing elements are connected with an 8-bit single-chip computer. Every sensor of this model is temperature compensated and calibrated in accurate calibration chamber, and the calibration-coefficient is saved in OTP memory. When the sensor is detecting, it will cite the coefficient from memory. Small size, low consumption and long transmission

distance(100m) enable AM2302 to be suited to all kinds of harsh application occasions.

Single-row packaged with four pins, making the connection very convenient.

Technical Specification:

- 3.3-5.5V DC Power supply
- Output signal: digital signal via 1-wire bus
- Sensing element: Polymer humidity capacitor
- Operating range: humidity 0~100 %RH; temperature -40~80 Celsius
- Accuracy: humidity ± 2 %RH(Max ± 5 %RH); temperature ± 0.5 Celsius
- Resolution or sensitivity: humidity 0.1 %RH; temperature 0.1 Celsius
- Repeatability: humidity ± 1 %RH; temperature ± 0.2 Celsius
- Humidity hysteresis: ± 0.3 %RH
- Long-term Stability: ± 0.5 %RH/year
- Interchangeability: fully interchangeable

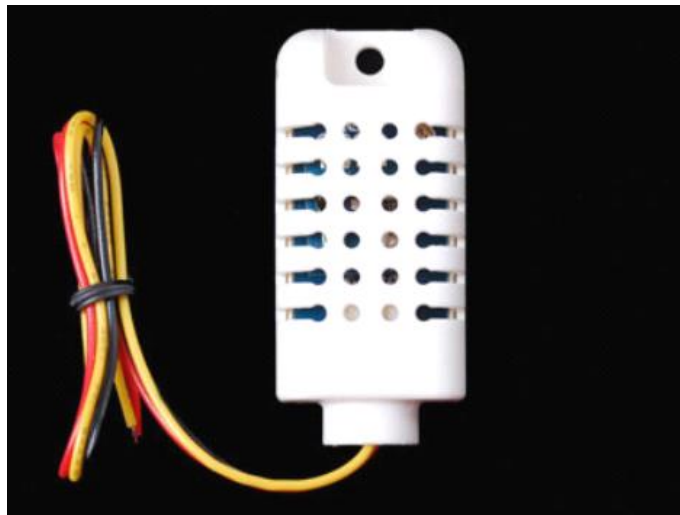


Figure 3.5: AM2302

3.2.1.3 BME280 -Temperature Humidity Pressure Sensor

To fulfill the requirement of small-size, we have chosen BME280 as the pressure sensor. BME280 is an environmental sensor with temperature, barometric pressure, and humidity. This sensor is excellent for all sorts of weather/environmental sensing and can be used in both I2C and SPI. This precision sensor from Bosch is a low-cost sensing solution for measuring humidity with $\pm 3\%$ accuracy, barometric pressure with ± 1 hPa absolute accuracy, and temperature with $\pm 1.0^{\circ}\text{C}$ accuracy. BME280 is the next-generation of sensors from Bosch and is the upgrade to the BMP085/BMP180/BMP183 - with a low altitude noise of 0.25m and the same fast conversion time. It has the same specifications, but developers can use either I2C or SPI.

Key features:

- Package 2.5 mm x 2.5 mm x 0.93 mm metal lid LGA
- Digital interface I²C (up to 3.4 MHz) and SPI (3 and 4 wire, up to 10 MHz)
- Supply voltage VDD main supply voltage range: 1.71 V to 3.6 V
- VDDIO interface voltage range: 1.2 V to 3.6 V
- Current consumption 1.8 μA @ 1 Hz humidity and temperature
- 2.8 μA @ 1 Hz pressure and temperature
- 3.6 μA @ 1 Hz humidity, pressure, and temperature
- 0.1 μA in sleep mode
- Operating range $-40\sim+85^{\circ}\text{C}$, 0~100 % rel. humidity, 300~1100 hPa

- Humidity sensor and pressure sensor can be independently enabled/disabled

Key parameters for humidity sensor:

- Response time 1 s
- Accuracy tolerance $\pm 3\%$ relative humidity
- Hysteresis $\pm 1\%$ relative humidity

Key parameters for pressure sensor:

- RMS Noise 0.2 Pa, equivalent to 1.7 cm
- Offset temperature coefficient ± 1.5 Pa/K, equivalent to ± 12.6 cm at 1 °C

temperature change

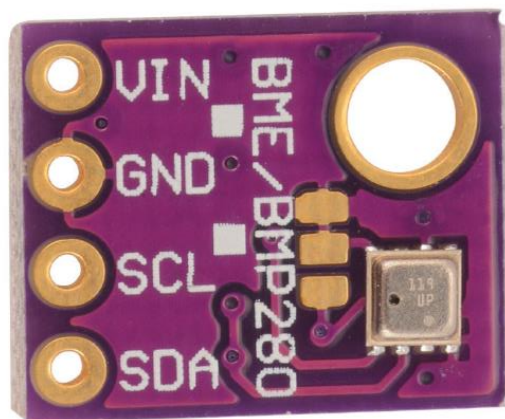


Figure 3.6: BME280

3.2.2 Boards

In this project, we use a single-board computer, Raspberry Pi Zero, and another RTC board DS3231 to keep track of time.

3.2.2.1 Raspberry Pi Zero

In the early stage of prototyping, we used Raspberry Pi model 3 as our single board computer. It has quad-core 1.2GHz Broadcom BCM2837 64bit CPU, which is pretty powerful for a single board computer, and 1GB of RAM on the board. It is also equipped with full-size HDMI, which is pretty handy during the development since we can use it to connect to an external monitor.

However, after we finished the software, we found that the excellent performance of this board is redundant in this project. The system will become expensive and consume much power if we use this board, which is the opposite direction of the original intention. To fulfill the design requirements of low-cost and low-power, we dropped this board and moved to Raspberry Pi Zero W.

The Raspberry Pi Zero W is a new small single-board computer from the Raspberry Pi Foundation from the UK. Its size is just $65\text{mm} \times 30\text{mm} \times 5\text{mm}$, as small as a piece of gum. So it perfectly meets the requirements of miniaturization of this project. Launched at the end of February 2017, the Pi Zero W has all the functionality of the original Pi Zero, but comes with added connectivity, consisting of:

- 802.11 b/g/n wireless LAN
- Bluetooth 4.1
- Bluetooth Low Energy (BLE)

Like the Pi Zero, it also has:

- 1GHz, single-core CPU
- 512MB RAM

- Mini HDMI and USB On-The-Go ports
- Micro USB power
- HAT-compatible 40-pin header
- Composite video and reset headers
- CSI camera connector¹⁷



Figure 3.7: Raspberry Pi Zero W

3.2.2.2 DS3231 Real time clock

The DS3231 is a low cost, high precision I2C real-time clock (RTC) with an integrated temperature-compensated crystal oscillator (TCXO) and crystal. The device includes a battery input that maintains accurate timing when the mains supply is disconnected. The integrated crystal oscillator increases the long-term accuracy of the device.

The RTC saves seconds, minutes, hours, weeks, dates, months, and years. For months less than 31 days, the final days of the month will be automatically adjusted. It

also corrects the days for the leap year. The working format of the clock can be 24 hours or a 12-hour format with AM/PM indication. Two configurable calendar alarms and a configurable square wave output are available. The address and data are transmitted serially via the I2C bidirectional bus.

A precision, temperature-compensated voltage reference, and comparator circuit monitors the VCC status, detects power failures, provides a reset output, and automatically switches to the backup supply when necessary. In addition, the RST monitor pin can be used as a manual input to generate a reset of the microprocessor.



Figure 3.8: DS3231 RTC

Key characteristics:

- Accuracy of $\pm 2\text{ppm}$ from 0°C to $+40^{\circ}\text{C}$
- Accuracy is $\pm 3.5\text{ppm}$ from -40°C to $+85^{\circ}\text{C}$
- Provide battery backup input for continuous timing

- Range of working temperature: Commercial grade: 0°C to +70°C
Industrial grade: -40°C to +85°C
- Low power consumption
- The real-time clock generates seconds, minutes, hours, weeks, dates, months, and years, and provides leap year compensation that is valid until 2100
- Two calendar alarm clocks
- Programmable square wave output
- High speed (400kHz) I2C interface
- Working at 3.3V
- Digital temperature sensor output: $\pm 3^{\circ}\text{C}$ accuracy
- Ageing correction register
- RST output/button reset debounce input

3.2.3 Casing

For the implemented hardware system, we have designed and built a case using 3D printing, which is mainly designed for protecting the hardware from environmental noise (sunlight, water, etc.). The implemented sensing system is shown in Figure 3.9.

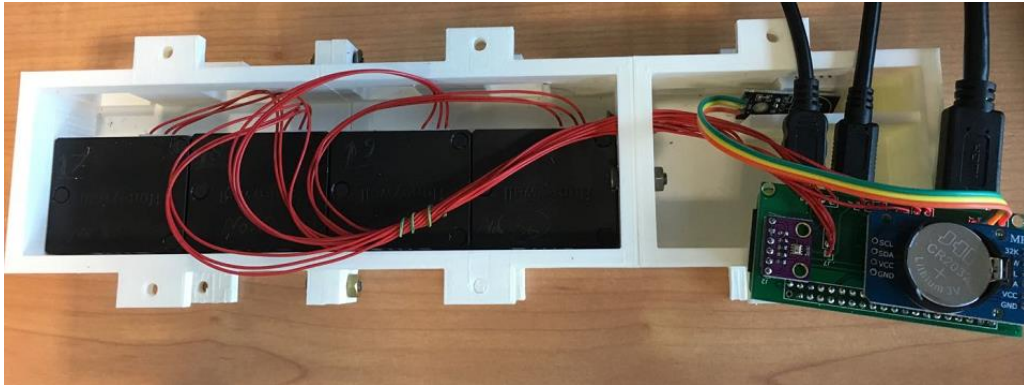


Figure 3.9: one platform with case

3.3 Software

As the requirements of the project involved, some requirements for software have to be fulfilled to make the final product more acceptable by the market. They are:

- The operating system should be lightweight (low-power);
- The operating system and other software should be open-source or free to use (low-cost);
- It is better to choose popular software or languages since it would be easier for prototyping;
- Software for database and user interface should be able to deal with massive amount of data.

As the final platform should be low-power, stable and networkable, it is also crucial to carefully choose an appropriate operating system and programming language(s).

The Arch Linux ARM was chosen as the operating system for this platform because it is a clear Linux system without redundant packages and libraries. Using this lightweight OS further decreases the power consumption of the platform.

C was chosen as the primary programming language for the communication between the board and sensors. There are several reasons for doing so:

- Portable, it makes the possible transplant after prototyping easier;
- High efficiency, compared with Python or other high-level languages, C is more efficient which helps to meet the low-power requirement;
- Widely used, there are many reusable open-source projects written in C;
- Stable, it is crucial to have the ability to operate continuously.

After the design of the software structure to meet the requirements of the whole system, the software of this project is divided into 3 parts:

1. On Raspberry Pi Zero, there are a series of scripts and a C program which configure the environment, synchronize data with the server, initialize the sensors, and reads data periodically.

2. There are also two Python scripts running on a remote server. One Python script is for receiving data from boards and put them into a database where all the data are stored and ready to be read by users. Another Python script is for catching data of traffic flows in Turin from 5T (a website which provides services and innovation for mobility in Turin) and put them in a table of the database also waiting to be used.

3. A visualization interface made with Grafana for users to check air pollution or traffic situation on a map. It makes a query from the database and shows them on a user-friendly interface.

3.3.1 Raspberry Pi side

Software on Raspberry Pi is intended to get environment data and send them to the server periodically. For this purpose, we have designed a series of scripts which execute several different jobs.

Since we have 12 boards in the test station, we cannot synchronize data files on all board with the server at the same time (because it requires a relatively higher network speed). So the first script we have on Raspberry Pi is called "crontab_select.sh" which is used to decide what the crontab jobs are for different boards. The algorithm of this script is shown in Figure 3.10. We number the 12 boards in the test station as "0, 1, 2, ..., 10, 11" and save this id of each board in a .txt file. In the script, we first read this number from the file and decide which crontab we want to use.

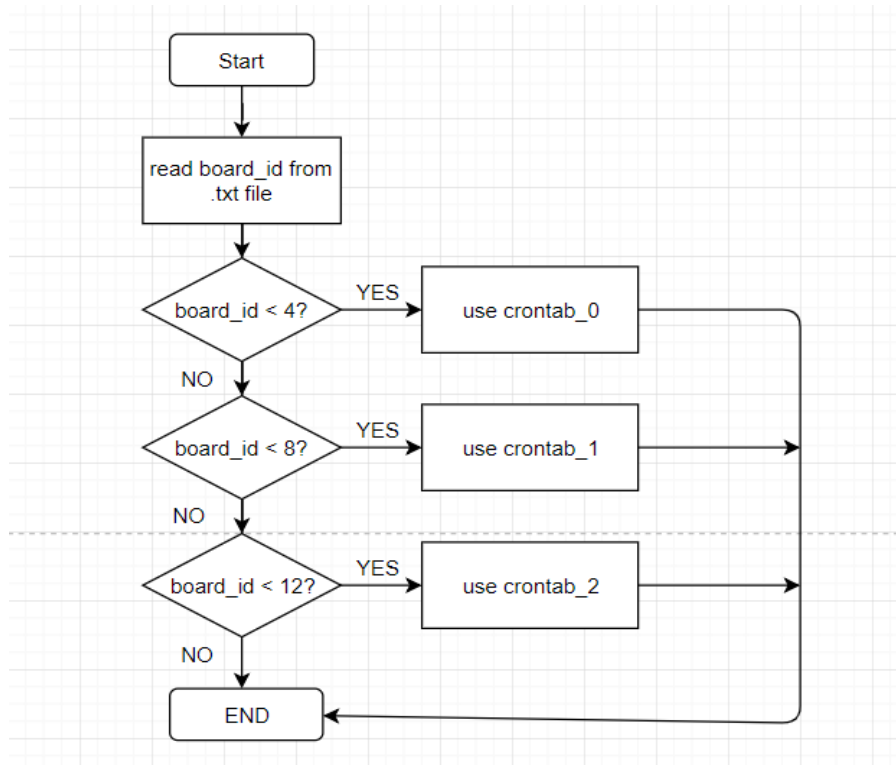


Figure 3.10: algorithm of "crontab_select.sh"

After executing "crontab_select.sh", the 12 boards are divided into 3 groups in which every 4 boards use the same crontab. For example, the content of "crontab_0" is shown in Figure 3.11.

```

1 @reboot chmod +x /home/alarm/sensing_station_platform/script/my_chmod.sh
2 @reboot /bin/bash /home/alarm/sensing_station_platform/script/my_chmod.sh
3 @reboot sleep 5 && timedatectl --adjust-system-clock && hwclock -s
4 @reboot sleep 5 && /bin/bash /home/alarm/sensing_station_platform/script/run.sh
5 @reboot sleep 5 && /bin/bash /home/alarm/sensing_station_platform/script/connect_qc.sh
6
7
8 15 8 * * 2,5 /bin/bash /home/alarm/sensing_station_platform/script/until_doomsdayPUSH.sh
9 15 10 * * 3,6 /bin/bash /home/alarm/sensing_station_platform/script/until_doomsdayPUSH.sh
10 15 15 * * 1,4,7 /bin/bash /home/alarm/sensing_station_platform/script/until_doomsdayPUSH.sh

```

Figure 3.11: crontab_0

In the crontab, there are several jobs after each reboot for setting up the environment and several scheduled jobs for pushing data to the server. The script "my_chmod.sh" grant execution permission to all the scripts in this project to avoid potential permission

problems when executing any script. Then after 5 seconds of sleeping (to make sure that the system is ready), we regulate system clock and run the script "run.sh" which compiles (if it is needed) and executes the c program for reading data from the sensor and save them to CSV files.

For the scheduled pushing, we assign different time for each group of boards to avoid the hotspot having too many devices connected at the same time. In this way, at set intervals, the script "until_doomsdayPUSH.sh" is executed for sending data to the server. There are 4 scripts executed in "until_doomsdayPUSH.sh".

The first which is called "connect_update_time.sh" is intended to connect the board to an available hotspot and the algorithm of it is described in Figure 3.12. As it is shown, we use 3 hotspots for this test station in case of the failure of any single smartphone. The board will try these 3 hotspots one by one, and it will try to push again at next set time if all the hotspots are not available.

The second script in "until_doomsdayPUSH.sh" is "my_mv.sh" which move all the ready-to-synchronize files to a folder waiting to be sent. After that is finished, the third script "my_rsync.sh" will call "rsync" (an open source utility that provides fast incremental file transfer¹⁸) to synchronize all the data files with a remote server.

The last executed script is called "disconnect_wlan0.sh". This script will tell the board to disconnect from the hotspot. By doing this, the power consumption of the board is further decreased.

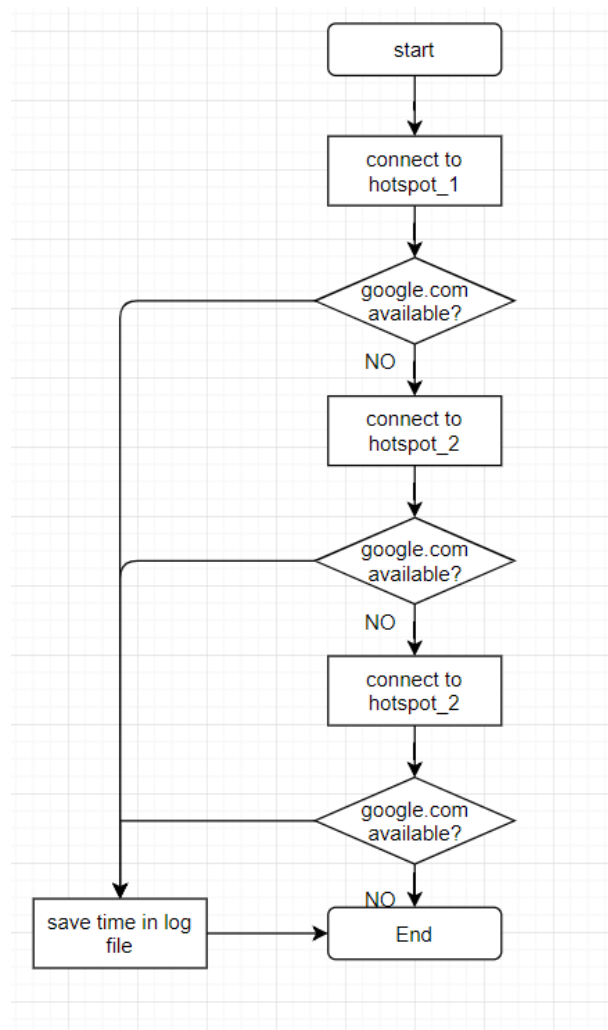


Figure 3.12: algorithm of "connect_update_time.sh"

Now let us further analyze the code running on the Raspberry Pi zero for getting and saving data from sensors.

The PM sensors use UART (Universal Asynchronous Receiver Transmitter) to transfer data, but there is only 1 physical UART on the Raspberry Pi Zero W (one pin for TXD and another for RXD). Therefore, we must use a software UART. By using the library "pigpio" which is developed for Raspberry Pis to manipulate the GPIOs¹⁹, 4 GPIOs are taken as input ports for 4 software UARTs.

The auto-send mode of this PM sensor is enabled by default after powered up. What Raspberry Pi needs to do is to read in the data from different pins and check them according to the data format given by the manufacturer.

Besides, the relative humidity / temperature sensor AM2302 uses the 1-wire bus with a particular communication process (shown in Figure 3.13), so I have implemented a specific c program for communicating with it. Also, any GPIO could be used to do this job since it does not ask for a particular protocol.

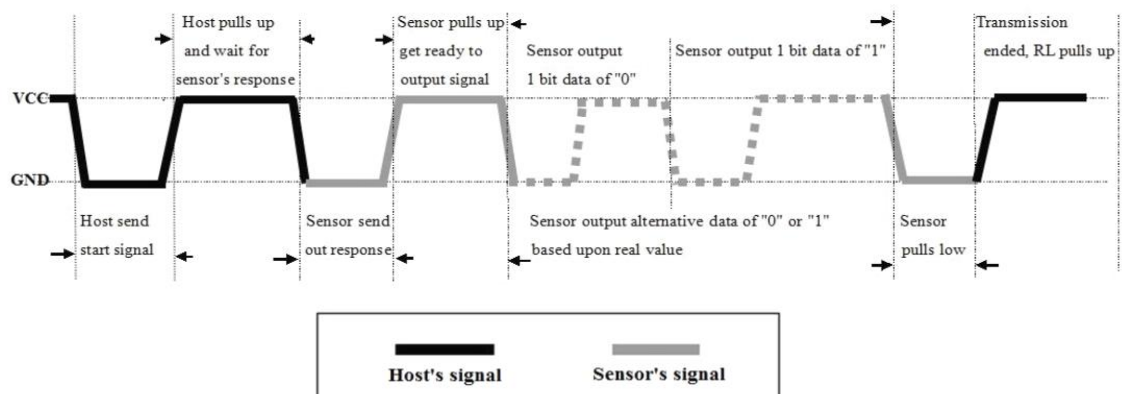


Figure 3.13: communication process of AM2302

As for the sensor BME280, I have also written a code segment which is dedicated to reading data from it by using the I2C pins on Raspberry Pi.

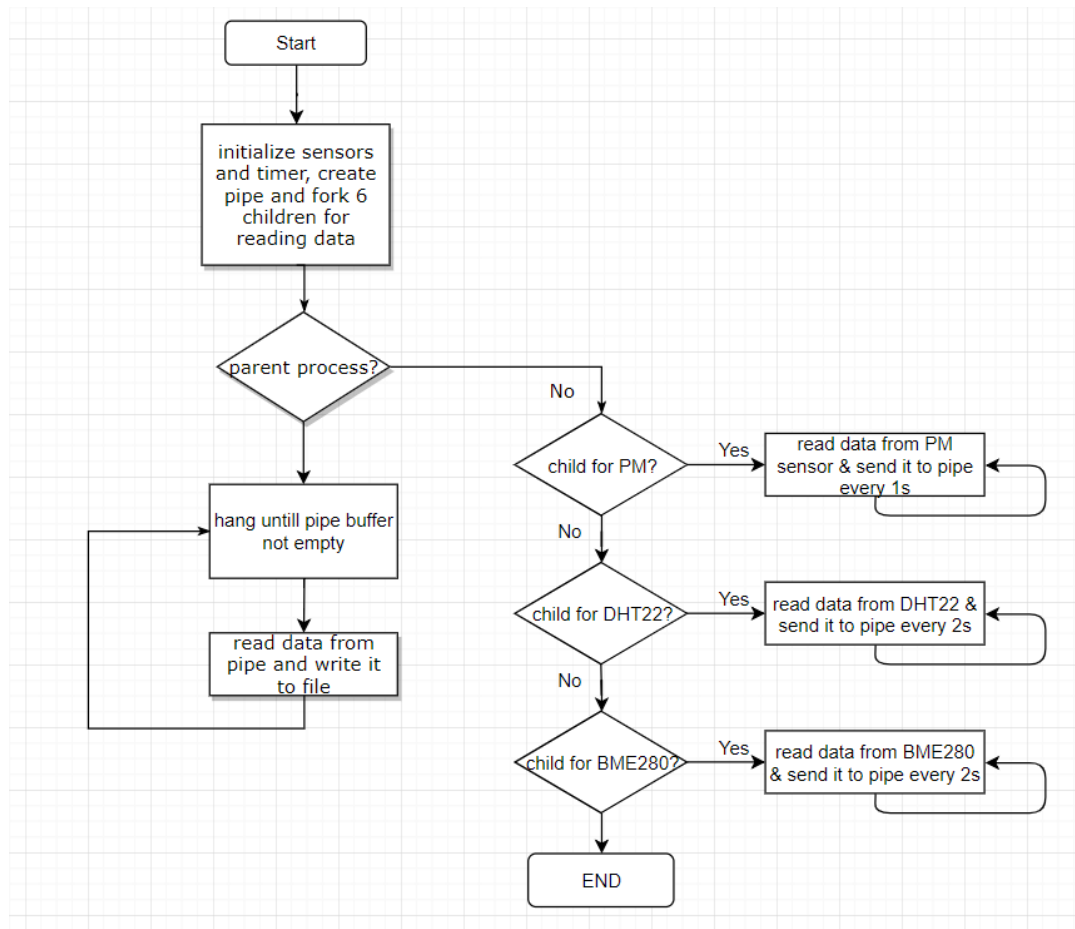


Figure 3.14: flowchart of code for sensors on Raspberry Pi Zero

Now let us discuss the architecture of code for sensors. In this C program, I use one parent process to manage the infinite loop of reading data from child processes and write them to disk. And there are six child processes, each of which corresponds to a sensor. One child process can read from one sensor using a specified protocol.

At first, the parent process initializes a pipe which is used for data transferring later. After doing that, it forks 6 child processes. Then the child processes start to do the work about the sensors. The parent process will also set up an alarm which rings every one minute. As it is shown in Figure 3.15, the parent process will make a check for the death

of every child processes everytime this alarm rings. If any child process is reported as dead, the parent process should restart it immediately.

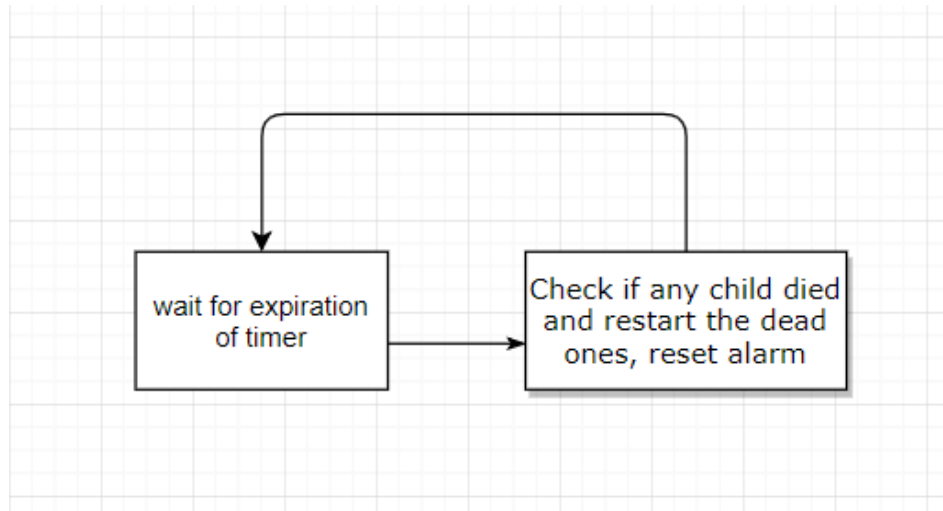


Figure 3.15: flowchart of the alarm

For the reading and saving, the parent enters an infinite loop in which it checks if any child has written something in the pipe. If yes, it should read and write it to a CSV file in a specified format.

3.3.2 Server side

The job of the server is to receive data from platforms or catch data from specified websites and display the data to users in a friendly interface.

For displaying the data from 5T, the main problem is that the data provided by 5T are in XML format. It is not easy to directly insert them into the database or use them in the user interface. So a better solution would be parsing data in the XML file and insert them into a database which would provide the data to users later. The language chosen for doing so is Python because there are excellent parsers in modules for Python.

I use a Python script to get data from the 5T website. In the code, it reads the data from an XML file given by 5t, parses them and put them into a database finally.

Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.²⁰ In this Python script, I have used the "xml" package for processing XML data. By using the methods in it, this script can parse the data to Python objects and then call methods in package "mysql" in order to insert data into the database.

The reason for choosing Python as the language for this job is that Python is widely used in many operating systems and there are many good modules ready to be used which makes the development much easier.

Besides, the data from 5T includes the location information in the form of longitude and latitude. But in order to show it on the map later, it is necessary to convert them to geohash (because it is easier to use in Grafana). I have used another module called "Geohash" to do the conversion from longitude and latitude to a geohash number.

Geohash is a public domain geocoding system invented by Gustavo Niemeyer, which encodes a geographic location into a short string of letters and digits. It is a hierarchical spatial data structure which subdivides space into buckets of grid shape, which is one of the many applications of what is known as a Z-order curve, and generally space-filling curves.²¹

```
mysql> select * from 5tdata;
```

start_time_UTC	end_time_UTC	lcd1	Road_LCD	Road_name	offset	direction	lat	lng	Geohash	accuracy	period	flow	speed
2018-09-10 20:05:00	2018-09-10 20:10:00	4856	335	SS23 Del Colle Di Sestriere	765	negative	44.9552	7.19947	spvpezf7pb9v	100	5	108	55
2018-09-10 20:05:00	2018-09-10 20:10:00	4851	335	SS23 Del Colle Di Sestriere	4257	positive	44.879	7.31392	spvpq3vz4xwm	100	5	276	82.88
2018-09-10 20:05:00	2018-09-10 20:10:00	4877	336	SS24 Del Monginevro	5066	negative	45.119	7.21835	u0j0sm0jrdzv	100	5	36	64
2018-09-10 20:05:00	2018-09-10 20:10:00	4876	336	SS24 Del Monginevro	1922	positive	45.119	7.21835	u0j0sm0jrdzv	100	5	12	89
2018-09-10 20:05:00	2018-09-10 20:10:00	40182	40180	Corso Unione Sovietica(TO)	434	positive	45.0164	7.62801	u0j2jdfx9e09	100	5	300	46.8
2018-09-10 20:05:00	2018-09-10 20:10:00	40183	40180	Corso Unione Sovietica(TO)	728	negative	45.0179	7.62949	u0j2je78kn2h	100	5	312	24.11
2018-09-10 20:05:00	2018-09-10 20:10:00	40182	40180	Corso Unione Sovietica(TO)	707	positive	45.0182	7.63031	u0j2jek4r05t	100	5	372	21.26
2018-09-10 20:05:00	2018-09-10 20:10:00	40182	40180	Corso Unione Sovietica(TO)	1007	positive	45.0203	7.63267	u0j2jetyq05q	94	5	156	17.33
2018-09-10 20:05:00	2018-09-10 20:10:00	40095	40093	Corso Orbassano(TO)	710	positive	45.0311	7.61203	u0j2jjwvnhuv	100	5	552	64.43
2018-09-10 20:05:00	2018-09-10 20:10:00	40312	40309	Via Vigliani(TO)	64	negative	45.0224	7.63587	u0j2ju062xtm	78	5	36	30.85
2018-09-10 20:05:00	2018-09-10 20:10:00	40183	40180	Corso Unione Sovietica(TO)	42	positive	45.023	7.63569	u0j2ju0ju7z1	100	5	372	33.68

Figure 3.16: 5t data in the database on the server after parsing in Python

As for the database, in this project, I have used MySQL as the database management system. The reason for choosing it is that MySQL is open-source and good at big data. For a massive amount of data, usually, we use a relational database management system (RDBMS) to store and manage them. It is called a relational database because all data is stored in different tables. The relationship between the tables is based on the primary key or other keys (called foreign keys).

MySQL is a fast-to-use RDBMS that many businesses (regardless of size) use to build their own databases. MySQL was developed, operated and supported by a Swedish company, MySQL AB.

There are 5 tables in the database "weather_station" (the database used to store all data related to sensor nodes). Their explanations are as follows:

1. Table "MEASURE_TABLE" is used to store the measured data (numbers of PM2.5, temperature, etc.) with a timestamp.
2. Table "SENSOR_TABLE" is used to store all the data related to each sensor (ID, kind, unit, etc.).

3. Table "BOARD_TABLE" is filled with information for boards (IP, version of the software, etc.).
4. Table "POSITION_TABLE" is filled with position information of each sensor (longitude, latitude, and latitude).

The structure of this database is shown as the following figure.

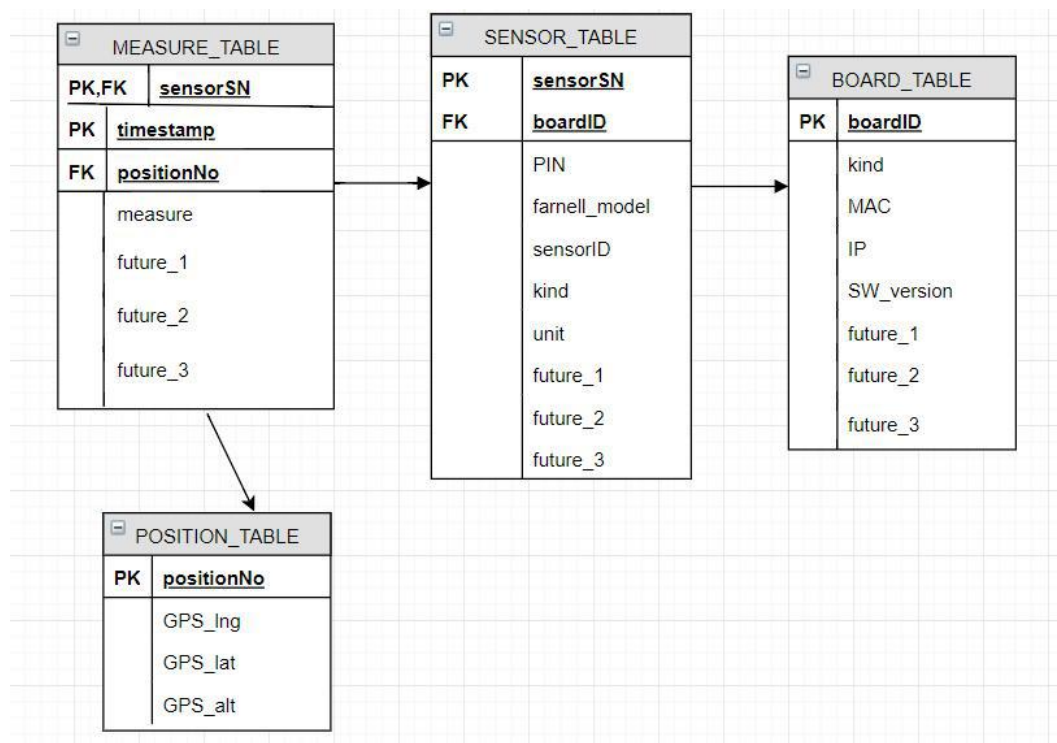


Figure 3.17: structure of weather station database

In the figure of database structure, "PK" is the abbreviation for "primary key." The primary key is an attribute or group of attributes that uniquely identifies a row in a table. A table can only have one primary key but can have multiple candidate indexes. Primary keys often form a referential integrity constraint with foreign keys to prevent data inconsistencies. The primary key ensures that the data is unique.

And "FK" is short for "foreign key." A foreign key is one or more columns used to establish and enforce a link between two tables. Foreign key constraints are mainly used to maintain the consistency of data between the two tables. In short, the foreign key of the table is the primary key of another table, and the foreign key links the two tables. In general, to delete a primary key in a table, you must first ensure that there are no foreign keys in any other table (that is, the primary key in the table does not have a foreign key associated with it).

For implementing the database on the server, I have written a script in which a series of MySQL commands are included. A server administrator can easily create databases for this project by executing this script. A part of it is shown in Figure 3.18.

```
-- run this script to create database
-- and tables for weather station data
-- example : "mysql -u emilio -p < /home/ala
CREATE DATABASE WEATHER_STATION;
USE WEATHER_STATION;
-- creat table for measurement data
-- sensorSN is logical ID of sensor
CREATE TABLE `MEASURE_TABLE`
(
  `sensorSN` INT NOT NULL,
  `timestamp` DATETIME NOT NULL,
  `positionNo` INT NOT NULL,
  `measure` FLOAT NOT NULL,
  `future_1` FLOAT NULL,
  `future_2` FLOAT NULL,
  `future_3` FLOAT NULL,
  PRIMARY KEY(sensorSN,timestamp)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
-- creat table for position
CREATE TABLE `POSITION_TABLE`
(
  `positionNo` INT NOT NULL,
  `GPS_lng` FLOAT NULL,
  `GPS_lat` FLOAT NULL,
  `GPS_alt` FLOAT NULL,
  PRIMARY KEY(positionNo)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
-- creat table for sensor information
-- sensorID is physical ID of sensor
-----
```

Figure 3.18: script piece for creating the databases

3.3.3 Visualization interface

As the ultimate purpose of this project is monitoring the air pollution, it is critical to design and develop a good interface for monitoring.

In this project, Grafana has been used as the visualization tool. In order to fulfill the requirements of the system, there are several reasons for choosing Grafana:

- **Grafana is an open source program(low-cost);**
- **It is designed for visualizing large-scale measurement data;**
- **It also supports main popular data sources such as Graphite, InfluxDB, OpenTSDB, Elasticsearch, MySQL and so on;**
- **It is the leader in this area, so many resources are available.**

As it is shown in Figure 3.19, after the query and some computation, we can get some curves showing air quality parameters which update in real time.



Figure 3.19: dashboard of real data displaying

```
SELECT
  UNIX_TIMESTAMP(UTctime) as time_sec,
  `PM2d5` as value,
  'sensor1' as metric
FROM weather
WHERE $__timeFilter(UTctime) AND sensorNUM=1
ORDER BY UTctime ASC
```

Format as Time series Show Help ▶

Figure 3.20: query example(PM2.5) in Grafana

Except for the graphs, several panels have also been included in the dashboard in the form of maps. Especially for the traffic data from 5t, it is better to show them on a map because in this way users will have a direct perception of the data in real time. An example of the map is shown in Figure 3.21.

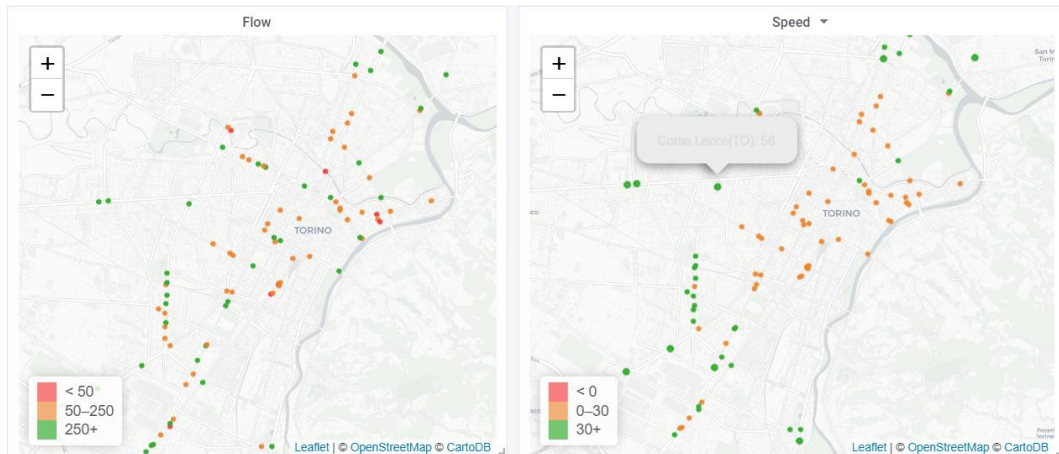


Figure 3.21: traffic data in real time on maps

Chapter 4 Experimental Data Analysis

4.1 Correlation Analysis of PM Sensors

Since there are 4 PM sensors in this project, the data from different sensors may conflict with each other. We need to test the correlation between different sensors before we use any of them.

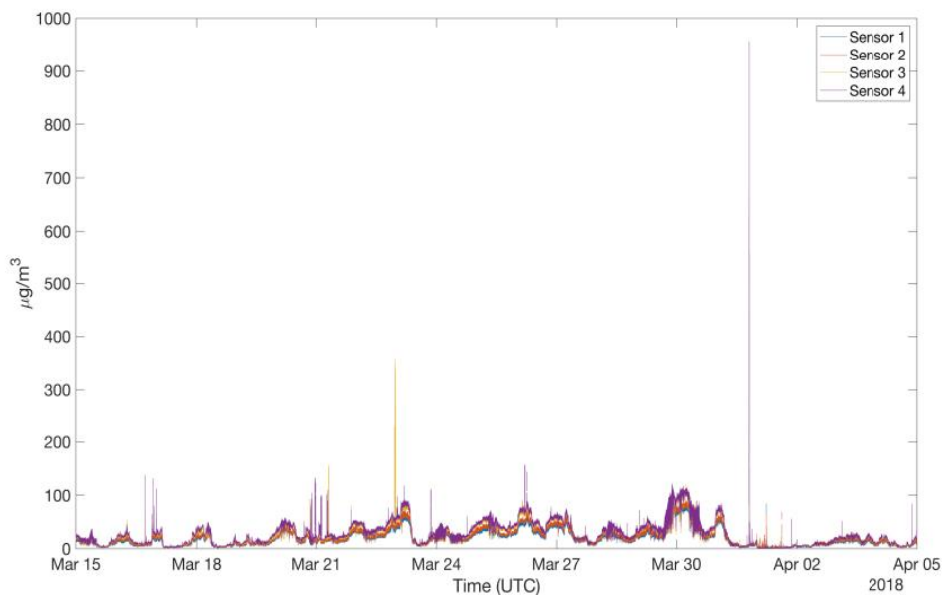


Figure 4.1: PM10 over 20 days

Figure 4.1 shows the concentration-time curve of PM10 detected by all 4 sensors. We can see that there are some peaks may happen to be noise. So we decided to apply a median filter on the data. This median filter was applied in Matlab using the function `medfilt1()`. The results are shown in Figure 4.2.

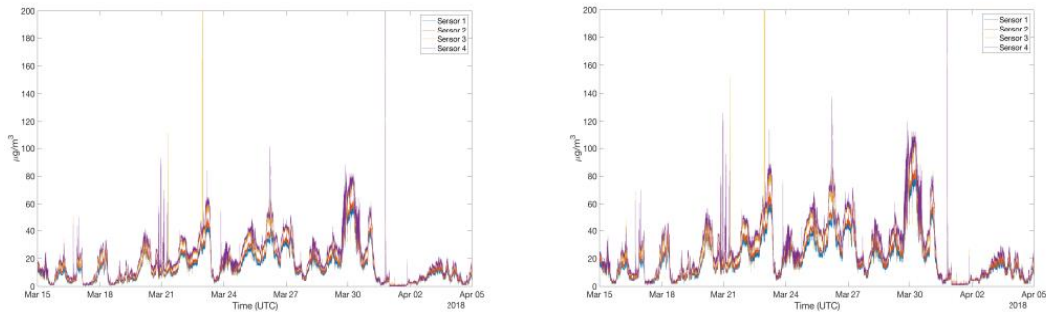


Figure 4.2: PM2.5 and PM10 concentration after median filtering

With the median filter applied, we can compute the correlation between different sensors. In Matlab, there is also another function for that which is called "plotregression()." By using this function, we can get the linear regression factor for each couple of PM sensors used in this project. In Figure 4.3 the correlation between sensor1 and sensor 2 in which we can see that they are highly correlated.

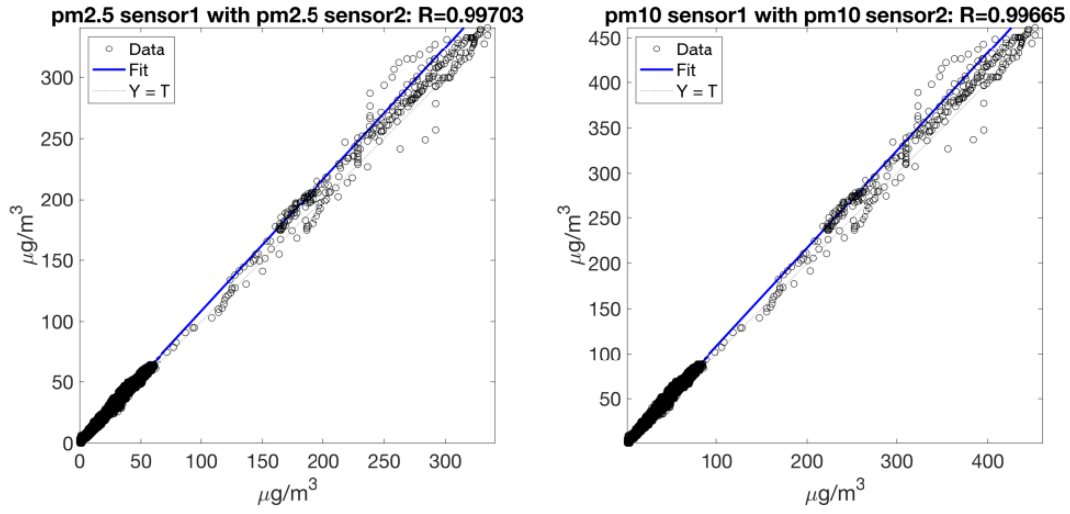


Figure 4.3: correlation between sensor 1 and sensor 2

$PM_{2.5}$	1-2	1-3	1-4	2-3	2-4	3-4
Raw data	0.993	0.949	0.983	0.950	0.986	0.952
Mean data	0.997	0.954	0.987	0.995	0.989	0.995
Median data	0.997	0.953	0.986	0.954	0.989	0.954

Table 4-1: PM2.5 correlation factors of sensor couples

In addition, PM2.5 and PM10 correlation factors between of each couple of the sensors are shown in Table 4-1 and Table 4-2. All of the factors are high, which proves that the sensors are consistent with each other.

PM_{10}	1-2	1-3	1-4	2-3	2-4	3-4
Raw data	0.993	0.949	0.982	0.951	0.985	0.953
Mean data	0.997	0.954	0.986	0.956	0.989	0.956
Median data	0.996	0.953	0.985	0.955	0.988	0.955

Table 4-2: PM10 correlation factors of sensor couples

4.2 Test of other sensors

For temperature and humidity, the case is more straightforward. Curves of samples are shown in Figure 4.4 which is also drawn in Matlab.

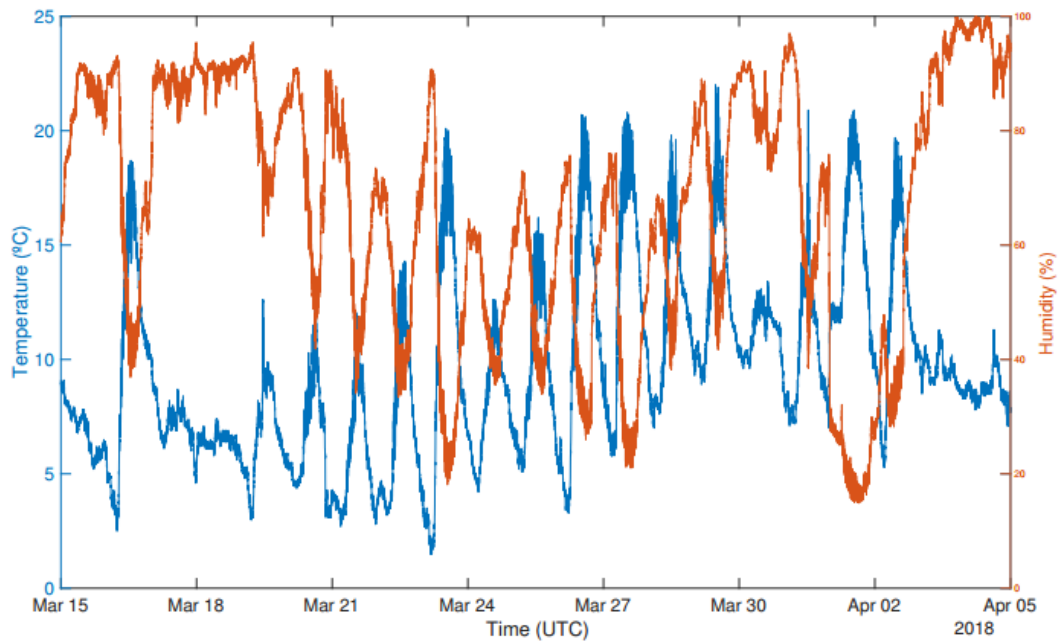


Figure 4.4: temperature and humidity samples

A piece of code to analyze the data in MATLAB is shown in Figure 4.5.

```
1 %read csv files as timetable
2 T_air_12=readtable('pin12.csv', 'Delimiter', ',', 'ReadVariableNames',true);
3 TT_air_12=table2timetable(T_air_12);
4 T_air_16=readtable('pin16.csv', 'Delimiter', ',', 'ReadVariableNames',true);
5 TT_air_16=table2timetable(T_air_16);
6 T_air_18=readtable('pin18.csv', 'Delimiter', ',', 'ReadVariableNames',true);
7 TT_air_18=table2timetable(T_air_18);
8 T_air_23=readtable('pin23.csv', 'Delimiter', ',', 'ReadVariableNames',true);
9 TT_air_23=table2timetable(T_air_23);
10
11 %combine all into a big timetable
12 TT_air_all=synchronize(TT_air_12,TT_air_16,TT_air_18,TT_air_23);
13
14
15 %plot figure for pm10 raw without truncation
16 fig0 =figure
17 set(gca,'FontSize',18)
18 set(gcf, 'Position', get(0, 'Screensize'));
19 fig0 = plot(TT_air_all.dateAndTime.UTC,TT_air_all.PM10_ug_m3__TT_air_12,TT_air_all.d
20 set(gca,'FontSize',18)
21 % title('This is my plot', 'FontSize', 18);
22 ylim([0 200])
23 xlabel('Time (UTC)', 'FontSize', 18);
24 ylabel('\mug/m^3', 'FontSize', 18);
25 set(gca,'FontSize',18)
26 legend('Sensor 1','Sensor 2', 'Sensor 3', 'Sensor 4');
27 print(gcf,'-depsc','pm10_raw_1000','-r500')
28
```

Figure 4.5: MATLAB script segment to process data

Chapter 5 Future Work

It should be noted that in this project we have still several points to be improved in the future.

First, we used a single board computer Raspberry Pi Zero W in this project which is more expensive and consumes more power than a traditional microcontroller used in IoT. So in the future, it is better if we transplant the software to a low-cost, low power microcontroller and design a PCB (which will also make the system more robust) for the new system.

Secondly, we are using a hotspot made by a smartphone as the internet access point. In order to make a real IoT node, we need to add another network module in the system (another solution is to use the one inside Raspberry Pi Zero W).

Now we are using a remote and local file synchronization tool called "rsync" to transmit data between Raspberry Pi and server. But this tool takes more time and

bandwidth compared with MQTT, a lightweight messaging protocol which is used for small sensors mobile devices all over the world.²² In order to achieve our goals for this system, we want to move to MQTT in the future.

Another thing is that we now only merely display all the data from sensors. There are chances that some of them are corrupted. So we need to write some code running in the server to process the data before visualizing them. In this way, we can avoid showing wrong data to the users.

Last but not least, we now have only a simple visualization interface for users who use PCs. As we all know, it is the mobile internet era now, so we need to write applications for mobile users in the future.

Chapter 6 Conclusion

This thesis is intended to propose a system which is designed to detect and monitor air pollution in modern cities. Motivations, ideas, architecture, and implementation of this project were explained in detail.

Proper sensors, boards, and tools have been used in this project to build a system which meets the design requirements and constraints.

The architecture of hardware and software is fundamental to the design of this air pollution sensing and monitoring system. What is equally important is the data visualization of each wireless sensor node in the grid. Hardware and software have been combined together in this project to serve the same goal.

A data visualization interface has been designed for the system, which is able to map urban air pollution on a large scale in real time.

The proposed system has been tested in real conditions by several experiments. Most requirements made in the design stage were met, and data analysis were done after the test.

In the future, after more specifications added to the project, a better system will be built and used in real conditions.

Conclusion

Bibliography

¹ ASHTON, Kevin. 2009. "That 'Internet of Things' Thing". *RFID Journal*.

<https://www.rfidjournal.com/articles/view?4986>.

² International Telecommunication Union. 2005. "ITU Internet Reports 2005: The Internet of Things". Geneva.

³Levis, Philip, Sam Madden, Joseph Polastre, Robert Szewczyk, Kamin Whitehouse, Alec Woo, David Gay, et al. 2005. "TinyOS: An Operating System for Wireless Sensor Networks." *Ambient Intelligence* 8491: 115–48. https://doi.org/10.1007/3-540-27139-2_7.

⁴ Levis, Philip, Sam Madden, Joseph Polastre, Robert Szewczyk, Kamin Whitehouse, Alec Woo, David Gay, et al. 2005. "TinyOS: An Operating System for Wireless Sensor Networks." *Ambient Intelligence* 8491: 115–48. https://doi.org/10.1007/3-540-27139-2_7.

- ⁵ "Micro-Controller Operating Systems". 2018. *En.Wikipedia.Org*.
https://en.wikipedia.org/wiki/Micro-Controller_Operating_Systems.
- ⁶ "Arch Linux - Archwiki". 2018. *Wiki.Archlinux.Org*.
https://wiki.archlinux.org/index.php/Arch_Linux.
- ⁷ "Particulates". 2018. *En.Wikipedia.Org*.
https://en.wikipedia.org/wiki/Particulates#cite_note-3.
- ⁸ Samet, Jonathan M., Francesca Dominici, Frank C. Curriero, Ivan Coursac, and Scott L. Zeger. 2000. "Fine Particulate Air Pollution And Mortality In 20 U.S. Cities, 1987–1994". *New England Journal Of Medicine* 343 (24): 1742-1749.
doi:10.1056/nejm200012143432401.
- ⁹ "Silent Tears". 2018. *Blog.Nus.Edu.Sg*. <http://blog.nus.edu.sg/silenttears/2017/09/>.
- ¹⁰ "Rooftop Sensors On U.S. Embassies Are Warning The World About ‘Crazy Bad’ Air Pollution". 2018. *Science | AAAS*. <http://www.sciencemag.org/news/2018/04/rooftop-sensors-us-embassies-are-warning-world-about-crazy-bad-air-pollution>.
- ¹¹ "China May Have Recorded Worst Pollution Ever". 2018. *That's Online*.
<http://www.thatsmags.com/china/post/11560/shenyang-pollution-soars-to-record-high-of-1-400>.
- ¹² "Air Pollution". 2018. *European Environment Agency*.
<https://www.eea.europa.eu/themes/air/intro>.
- ¹³ "Europe Pollution 'Kills 467,000 A Year'". 2018. *BBC News*.
<https://www.bbc.com/news/world-europe-38078488>.

- ¹⁴ "Wasmote - Open Source Sensor Node For The Internet Of Things | Zigbee, Sigfox, Lorawan, 3G / 4G Compatible | Libelium". 2018. *Libelium.Com*.
<http://www.libelium.com/products/wasmote/>.
- ¹⁵ Libelium. n.d. "Smart Gases PRO." *Power*, 1–42. www.libelium.com.
- ¹⁶ "Grafana - Archwiki". 2018. *Wiki.Archlinux.Org*.
<https://wiki.archlinux.org/index.php/Grafana>.
- ¹⁷ "Raspberry Pi Zero W - Raspberry Pi". 2018. *Raspberry Pi*.
<https://www.raspberrypi.org/products/raspberry-pi-zero-w/>.
- ¹⁸ "Rsync". 2018. *Rsync.Samba.Org*. <https://rsync.samba.org/>.
- ¹⁹ "Pigpio Library". 2018. *Abyz.Me.Uk*. <http://abyz.me.uk/rpi/pigpio/>.
- ²⁰ "XML". 2018. *En.Wikipedia.Org*. <https://en.wikipedia.org/wiki/XML>.
- ²¹ "Geohash". 2018. *En.Wikipedia.Org*. <https://en.wikipedia.org/wiki/Geohash>.
- ²² "MQTT". 2018. *Mqtt.Org*. <http://mqtt.org/>.