Advances in Technology Innovation, vol. 7, no. 1, 2022, pp. 01-18

Cost-Effective Customizable Indoor Environmental Quality Monitoring System

Mohammad Ayad Al-Rawi*, Praneel Chand, Archie Van Mendoza Evangelista

Centre for Engineering and Industrial Design, Waikato Institute of Technology, Hamilton, New Zealand

Received 17 August 2021; received in revised form 10 October 2021; accepted 11 October 2021

DOI: https://doi.org/10.46604/aiti.2021.8291

Abstract

Poor indoor environmental quality (IEQ) has become a global concern for World Health Organization (WHO), and its impact on health and well-being has been exacerbated by the COVID-19 pandemic. To monitor and sanitize indoor air, this study develops a cost-effective and customizable IEQ monitoring system to detect unhealthy and low-comfort air levels. This system uses ThingSpeak (MATLAB), microcontrollers (Arduino Uno), and various low-cost sensors to measure indoor air quality (IAQ) and IEQ in terms of gas, particulate matter, temperature, sound level, and ultraviolet (UV) light. The presented system is validated with respect to temperature, relative humidity, and particulate matter by benchmarking against the Camfil air image sensor manufactured by Camfil AB, Stockholm, Sweden. The average error of temperature, relative humidity, and $PM_{2.5}$ are 0.55%, 5.13%, and 3.45%, respectively.

Keywords: indoor environmental quality (IEQ), Internet of Things (IoT), ThingSpeak, sensors

1. Introduction

Due to the growth of industrial activities and the development of more intensive farming, the volume of carbon monoxide (CO), carbon dioxide (CO_2), and volatile organic compounds (VOCs) emitted has risen significantly over the past few decades [1]. Most people cannot detect these emissions (due to their odorless, colorless, and tasteless characteristics), nonetheless, they are harmful to health [2]. Colclough et al. [3] identified the increase in CO_2 level at night by using an indicator of the quality of ventilation in home. Indoor air can potentially contain a variety of harmful compounds, such as formaldehyde, dust and other allergens, radon gas, organic and inorganic materials, as well as water vapor which increases relative humidity (RH) and contributes to mould growth [4]. Poor indoor air quality (IAQ) due to the presence of VOCs [5] can be the result of the increased intensity of household use [3-4], or overcrowding, as well as the emissions from stoves and other cooking devices, heaters, and unflued fireplaces, such as CO, nitrogen oxide (NO), and sulphur dioxide (SO_2) [6].

In general, in developed countries, people spend 90% of their time indoors, with 70% of that time being spent in their own home [7]. The COVID-19 pandemic gave rise to an increase in these percentages: governments' enforced lockdowns resulted in an increase in remote working and schooling, conducted within home [8]. According to the work of Nwanaji-Enwerem et al. [8], 3.8 million people die from the diseases related to poor indoor air environment. Therefore, they advocate that Environmental Protection Agency must develop legislation regarding the integrity of the indoor air environment, to guarantee individuals a minimum level of protection, even in their own residential spaces.

Any mandate around indoor environmental quality (IEQ) would require comprehensive monitoring. However, such monitoring devices must be of low cost to be accessible for small-scale use, such as small business' offices and residential houses. Monitoring devices should have both thermal comfort applications as well as functionality to measure air quality or "safety". Devices should therefore measure particulate matter ($PM_{2.5}$), VOCs, CO, CO_2, and other compounds in the air. The COVID-19 pandemic has also increased the demand for detecting pathogens in the air. Recent studies [9-11] found a

* Corresponding author. E-mail address: mohammad.al-rawi@wintec.ac.nz

correlation between poor IAQ and the spread of viruses since 2020 [8]. To achieve a reliable and low-cost IAQ monitoring system, there are many different approaches presented in the literature, of the do-it-yourself (DIY) variety. Zhang et al. [12] used a low-cost Raspberry Pi-based system for the real-time monitoring of indoor environment parameters in both residential and commercial settings. They studied the correlation among $PM_{2.5}$, temperature (T), and RH, and noted that effective calibration of sensors required a reasonable length of run time and a comparison across different locations.

Recent studies indicated that the accuracy of the current low-cost monitoring systems for IAQ and IEQ is very important [13-17]. Therefore, the comparison between a low-cost sensor using a multi-channel monitoring system and calculated concentration is required. Additionally, the assessment of thermal comfort parameters (predicted mean vote (PMV) and predicted percentage of dissatisfaction (PPD)) is required in an advanced monitoring system. Parkinson et al. [18] investigated three innovative technologies, i.e., Internet of Things (IoT), wireless sensor networks (WSNs), and Big Data, to assess IEQ parameters using the SAMBA monitoring system. This system is used for the continuous and real-time data monitoring of IEQ categories in a commercial building. These categories are assessed against the ASHRAE Standard 55 for thermal comfort (T, air speed, RH, and radiant temperature), lighting, acoustical quality, and IAQ parameters (CO, CO_2, $PM_{2.5}$, and total VOCs). The SAMBA system is configured using Zigbee PRO mesh networking and uses the IEQAnalytics web service.

Previous studies used IoT wherein devices were embedded in a network using microcontroller sensors. These formed a web-based monitoring system which captured real-time data using Hypertext Transfer Protocol (HTTP) [19]. The advantage of using IoT is that it is a low-cost and reliable method for constructing a user-friendly IAQ monitoring system [5, 12, 20]. This platform is tested with pollution monitoring WSNs using IEEE 802.15.4 ZigBee [5-6, 12, 21-23]. Mumtaz et al. [11] employed IoT and machine learning to design a predictive IAQ system for parameters such as T, RH, $PM_{2.5}$, CO_2, CO, NO_2, and CH_4. Their system achieved an accuracy of 99.37% and precision of 99%.

Some studies showed how real-time IEQ readings and graphs could be presented on readily available smart phones [20, 22-23]. Taştan et al. [22] created a DIY system using the Blynk platform on an Android ESP32 module. Their IAQ monitoring system ran using the IoT method and presented real-time data to measure T, RH, CO_2, CO, PM_{10}, and NO_2. Marques et al. [20] presented a smartphone and ThingSpeak interface to capture IEQ data. Their system "iAir" used ESP8266 and MICS-6814. This enabled users to adjust devices from their smartphones in real time to improve IEQ. Wall et al. [23] built an IEQ monitoring system using the IoT architecture. Their system measured airborne compounds and other pollutants that impact the respiratory system, T, RH, and VOCs, using Raspberry Pi, EPSP32 microcontroller, IEEE 802.11 wireless local area network (LAN) router, and Bosch BME689 sensor [24-25]. Previous studies of monitoring systems also examined new approaches to combine deep learning and IoT to monitor air conditions in industrial settings and enhance energy conservation. The deep learning approach was based on a recognition algorithm to detect the number of occupants in the interest area, such as commercial or domestic buildings [26-29]. Therefore, their proposed approaches are considered industry 4.0 applications, relying on IoT and smart sensors to address industrial IEQ and noise signals using machine learning algorithms [26-29].

This study aims to create and validate a low-cost air quality monitoring system to detect whether the IEQ parameters, such as T, RH, PM_1, $PM_{2.5}$, VOC, CO, CO_2, ultraviolet (UV), and sound level, are aligned with National Environmental Standards for Air Quality Regulations in New Zealand [30]. In this work, cost-effective sensors are utilized by choice of low cost device components and cost of the overall system connected through an Arduino Uno microcontroller as WSNs, to monitor air pollutants in a residential dwelling using IoT as one of the technological innovations [18]. The measured data is visualized through an Android smart device or a PC linked to the sensor which aggregates, displays, and analyses real-time data in the cloud using ThingSpeak [31] as Big Data requires a business intelligence and analytics (BIA) platform to provide data visualization [18]. The ThingSpeak service is operated by MathWorks and can be obtained at zero-cost for small non-commercial projects. ThingSpeak includes a web service (REST API) that enables collection and storage of sensor data in the cloud for IoT applications.

2. The System Design

2.1. Hardware system architecture

The architechture of environmental monitoring system using the IoT-web based hardware is shown in Fig. 1. The proposed system consists of the following. (i) A particulate concentration sensor (PMS5003) is used to detect micro suspended particles in the room with the specifications outlined in Table 1. PMS5003 is supplied by a voltage of 3 V and a RX and TX connection from Arduino Uno to measure the particles in the air ($PM_{2.5}$). (ii) An IAQ sensor (MQ-135) is used to measure VOCs, NH_3, NOx, alcohol, Benzene, smoke, and CO_2 in the room with the specification shown in Table 2. MQ-135 requires a 5 V supply from Arduino Uno to transmit the VOC and gas readings. (iii) T and RH sensor (DHT-11) is used as shown in Table 3. The DHT-11 requires 3-5 V for sensing the T and RH of the monitored area. (iv) A UV sensor (XC4518) and a sound sensor (XC4438) are used as shown in Table 4 and Table 5, respectively. XC4518 and XC4438 are connected to a microcontroller (Tensilica 32-bit RISC CPU Xtensa LX106). These sensors are battery powered using the microcontroller (Arduino Uno and ESP2866 Wi-Fi microchip) communicated as a gateway to ThingSpeak for capturing real-time data and illustrating them as graphs which could be exported as comma-separated values (.csv) file under MathWorks. The captured graphs are used to display the real-time series for each sensor reading stored in the cloud database. Values can be displayed every 10 seconds, whereas the time series values are updated every minute showing all data updates on the dashboard. csv is used for the data, meaning the web application can send and retrieve the data from the server asynchronously without interfering with the displayed page.

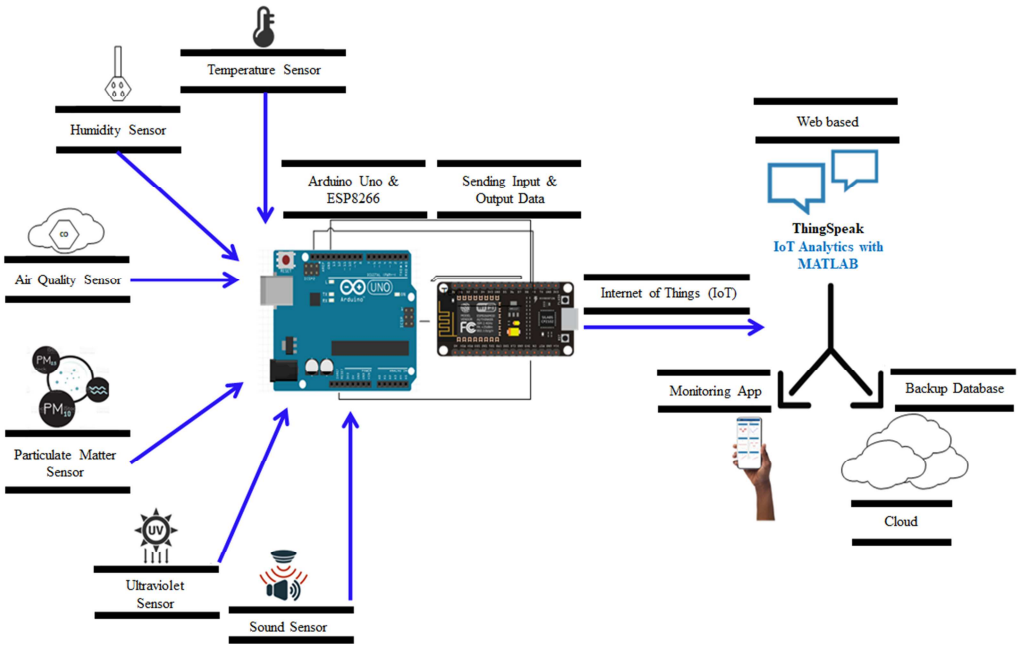


Fig. 1 The schematic diagram for the system architecture

Table 1 Specifications for particle concentration sensor (PMS5003)

Parameter	Index	Unit
Range of measurement	0.3-1.0; 1.0-2.5; 2.5-10	μm
Counting efficiecny	50%@0.3 98% @>=0.5	μm
Effective range	0-50	$\mu g/m^3$
Maximum range ($PM_{2.5}$ standard) Resolution	≥ 1000 1	$\mu g/m^3$ $\mu g/m^3$
Maximum consistency error ($PM_{2.5}$ standard)	$\pm 10\%@100-500$ $\pm 10@0-100-500$	$\mu g/m^3$ $\mu g/m^3$
Direct current power supply	Typ: 5.0; Min:4.5; Max: 5.5	Volt (V)
Physical size	50 × 38 × 21	Millimeter (mm)
Range of measurement	0.3-1.0; 1.0-2.5; 2.5-10	μm

Table 2 Specifications for air quality sensor (MQ-135)

Parameter	Index
Operating voltage	+5 V
Detect/measure	NH_3, NOx, alcohol, Benzene, smoke, CO_2, etc
Analog output voltage	0-5 V

Table 3 Specifications for temperature and humidity sensor (DHT-11)

Parameter	Index
Printed circuit board (PCB) size	22.0 mm × 20.5 mm × 1.6 mm
Working voltage	3.3 or 5 V direct current
Operating voltage	3.3 or 5 V direct current
Measurment range	20-95%RH; 0-50℃
Accuracy	±0.5℃; ±2%RH
Resolution	8-bit (T); 8-bit (RH)
Compatible interfances	2.54 3-pin interface and 4-pin

Table 4 Specifications for ultraviolet sensor (XC4518)

Parameter	Index
Wavelength response	200-370 nm
Protocol	Analog: 0-1.2 V direct current
Output voltage	0-1200 mV
Working temperature	-20 to 85℃
Current	0.06-0.1 mA
Supply voltage	3-5 V direct current
Dimensions	43 (L) × 13 (W) × 8(H)

Table 5 Specifications for sound sensor (XC4438)

Parameter	Index
Sensitivity	Adjustable via trimpot
Operating voltage	0-5 V direct current (analog)
Supply voltage	5 V direct current
Dimensions	43 (L) ×16 (W) × 13 (H)

One of the limiting factors in the initial proposed monitoring system is that a “blip” is observed due to the different voltage supply required for each sensor for data transfer. To fix that, a four-channel bi-directional logic level converter (LLC) (SparkFun) is used for the variance and to accommodate the variation of the voltage supply to each sensor from the microcontroller. The SparkFun logic device is designed to safely operate on the same channel as it steps up from the 3.3 V signals to 5 V and vice versa. Also, it works for future sensors within the range of 2.8 V and 1.8 V. The SparkFun logic device can convert 4 pins on the high side to 4 pins on the low side with two inputs and two outputs provided for each side. In addition to the bi-directional option of the SparkFun LLC, the board is easy to use, and is powered from the low voltage 3.3 V to “LV” and grounded from the system to the “GND” pin. Fig. 2 illustrates the final design. Fig. 3 shows the final product, and Table 6 shows the components of the project for each part connected to the microcontrollers (Arduino Uno and ESP8266) which are the core of the hardware and also the gateway to the IoT as well as supplying the required voltage to each of the sensors used in the system.

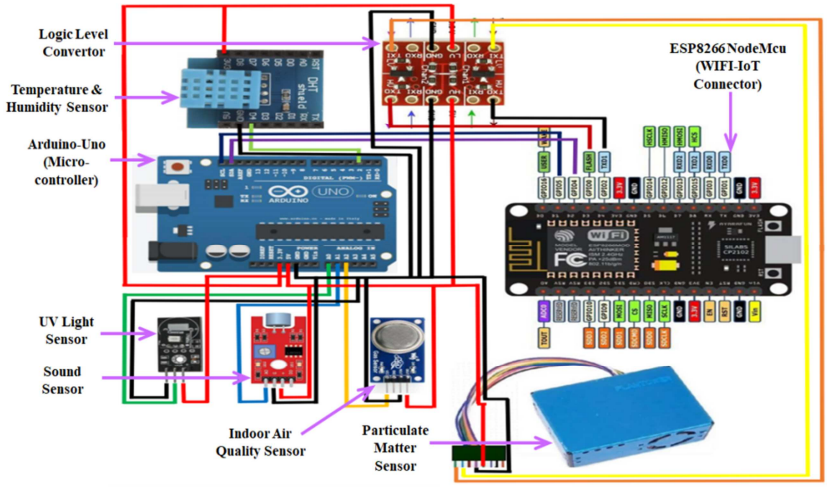


Fig. 2 Hardware setup illustrating the five sensors PMS5003, MQ-135, DHT-11, XC4518, and XC4438 along with the Arduino Uno and ESP8266 microcontroller unit and the IoT connector

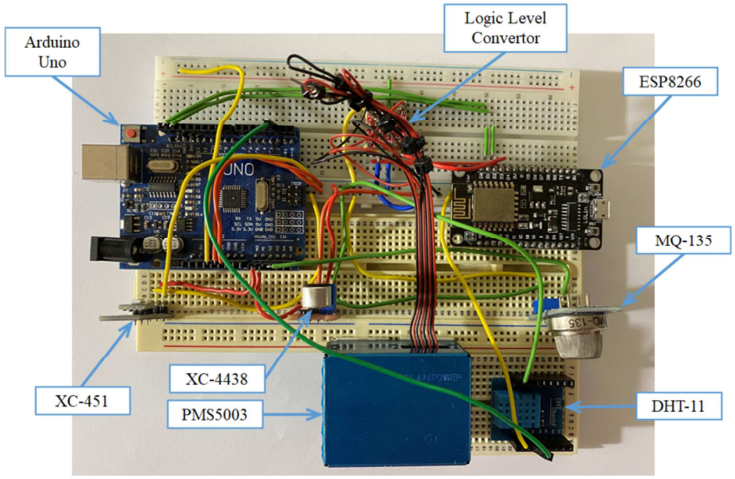


Fig. 3 Finished prototype of the IoT device

Table 6 The components of current design using IoT

Components	Specifications
Arduino Uno ESP8266 NodeMCU V3 Wi-Fi module	- Arduino Uno Microcontroller: Microchip ATmega328P Digital I/O pins: 14 (of which 6 can provide PWM output) Flash memory: 32 KB Operating voltage: 5 Volts - ESP8266 Wi-Fi module Microcontroller: Tensilica 32-bit RISC CPU Xtensa LX106 Digital I/O pins: 16 Flash memory: 4 MB Operating voltage: 3.3 V Clock speed: 80 MHz
Sensors and gateway (IoT)	- PMS5003 (Table 1) - MQ-135 (Table 2) - DHT-11 (Table 3) - XC4518 (Table 4) - XC4438 (Table 5)
End user	- Android and iOS systems - Laptop/desktop that can access to the Internet

2.2. Sensor calibration

The PMS5003, MQ-135, DHT-11, XC4518, and XC4438 sensors have their own required supply voltages. That is why the bi-directional LLC is needed to change the supply from 3.3 V to 5 V and vice versa on each cycle, and that is also the reason that there is a “blip” on the system reading as mentioned previously. The microcontrollers Arduino Uno and ESP8266 are tested and calibrated as shown in Fig. 4. This is to check that the right voltage and re-set/clock time is being supplied from the microcontrollers to the sensors and to ensure that the supply and ground are working in accordance with the required datasheet parameters.

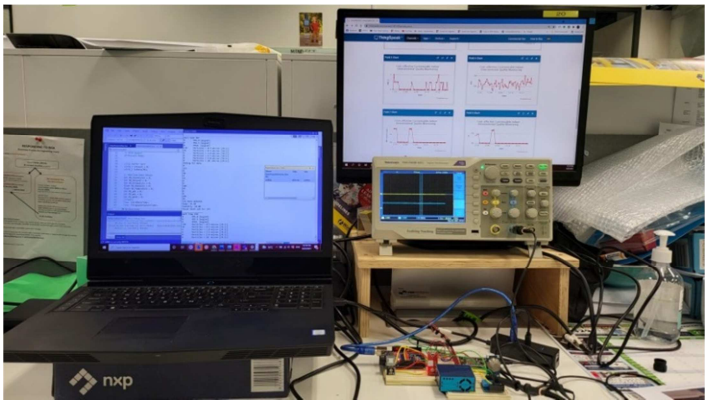


Fig. 4 Testing and calibration of the microcontroller using oscilloscope

2.3. *Software system architecture*

All the sensor data (except for the particulate sensor data) are read and collected by the Arduino Uno microcontroller as shown in Fig. 1. The Arduino microcontroller is a slave that relays the collected data to the ESP8266 master microcontroller. In turn, ESP8266 is responsible for transmitting the acquired data to the web-based IoT client. ESP8266 also reads the particulate sensor data. Microsoft Visual Studio Integrated Development Environment (IDE) with the vMicro extension is used for programming.

Fig. 5 shows the flowchart for data collection by the Arduino Uno slave microcontroller. The UV, sound, and air quality are read as 16-bit analogue inputs. T and RH data from the DHT11 sensor are 16-bit floating point (8 for T and 8 for RH) and read via serial communication. When the data is requested by the master ESP8266, the analogue inputs are converted and inserted into a 14-byte uint8_t data send array. They fill the first six bytes. Converting the 32-bit floating T and RH data fills the remaining 8 bytes in the data send array. The 14-byte data send array is then transmitted to the master. The flowchart for data receiving and transmitting by ESP8266 is shown in Fig. 6. After initializing ports and variables, the sensor data is requested from the Arduino Uno slave microcontroller. The sensor data is then re-built into their original formats. Following this, the particulate sensor data are read from the serial port and the $PM_{2.5}$ and PM_{10} data are extracted. Finally, when the ThingSpeak update timer passes, all the sensor data is transmitted to the HTTP uniform resource locator (URL) client.

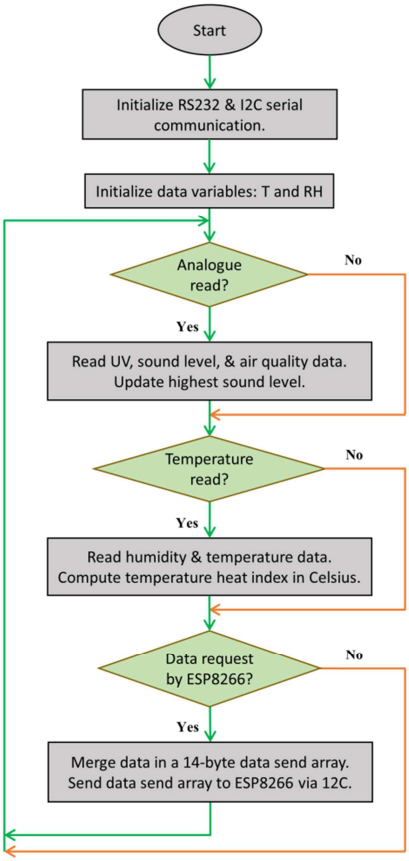


Fig. 5 The flowchart of data collection by the Arduino Uno slave microcontroller

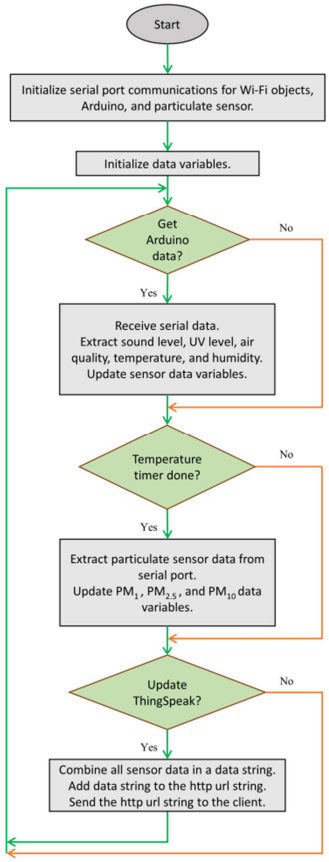


Fig. 6 The flowchart for data receiving and transmitting by ESP8266

3. Results and Discussion

A user-friendly and easy-to-assemble sensor can be constructed with interchangeable components enabling a low-cost device and using an open-source platform to capture and analyze the data for further validation and optimization [13]. Using the IoT technologies enables a wider range of people to be able to access and connect their devices to the sensor via the Internet, facilitating the collection and sharing of data. Wireless networks are nearly ubiquitous in most countries, benefitting the control and communication of the data.

The majority of the literature focuses on presenting a low-cost monitoring system, but in this study it is found that their proposed systems are often difficult to modify and lack modularity. Furthermore, some studies focus on the hardware, others on the software, but few concentrate on the both.

This study demonstrates the implementation of IoT low-cost and affordable approach with the use of ThingSpeak with five low-cost sensors to measure IAQ parameters, UV, and sound level. These sensors can be locally purchased in New Zealand and do not need sourcing from abroad thus eliminating procurement delays due to COVID-19 supply chain and distribution problems. A low-cost IoT solution could be significant for the task of monitoring, quantifying, and improving IAQ with the correct programming of each sensor. Furthermore, these sensors could be used in a smart home environment, where corrective action (e.g., opening a window or turning on an air purifier) could be triggered once the air quality drops below thermal comfort or IAQ recommended standards; this could mitigate the risk of illnesses due to poor IAQ.

The proposed "DIY" low-cost monitoring system in this study consists of three main components: the microcontrollers (the Arduino Uno and the ESP8266 microcontroller unit (MCU)) and the bi-directional LLC (SparkFun). The current system is easy to assemble and connect, and enables adding and modifying individual sensors in the system as technology advances. In addition, IoT is used via ThingSpeak by application programming interface (API) key under Matlab to capture the data which could be analyzed and optimized. To achieve that, the software is scripted using the files DataCollector program (Appendix A) to communicate to the Arduino Uno and the DataTransmitter program (Appendix B) and to the ESP8266 MCU and the IoT using Microsoft Visual Studio (vMicro extension) by IDE based on the architecture layers shown in Fig. 7. This allows for both code modification and error detection as it is separated on each MCU. Therefore, the key features of the proposed IAQ and IEQ monitoring system have the following advantages:

(1) The device can measure eight different ambient factors presented in the enclosed space: T, RH, CO_2, MP_1, $MP_{2.5}$, VOC, UV, and sound level.

(2) Due to the IoT, the device enables real-time monitoring and the analysis of different ambient impurities.

(3) The device does not require any secure digital (SD) memory card for data storage. All the data gathered is stored on the cloud sever and downloadable for future use.

(4) The device's modularity allows for the addition of more sensors, and the used sensors can be switched out for upgraded ones.

For the operation of both the DataCollector and DataTransmitter programs, a video is recorded available as supplementary material. The security of the captured data using WSN requires a username and password to access ThingSpeak. Additionally, the DataTransmitter program requires a secure Internet by setting and defining the Wi-Fi parameters WLAN_SSID and WLAN_PASS as shown in Fig. 8.

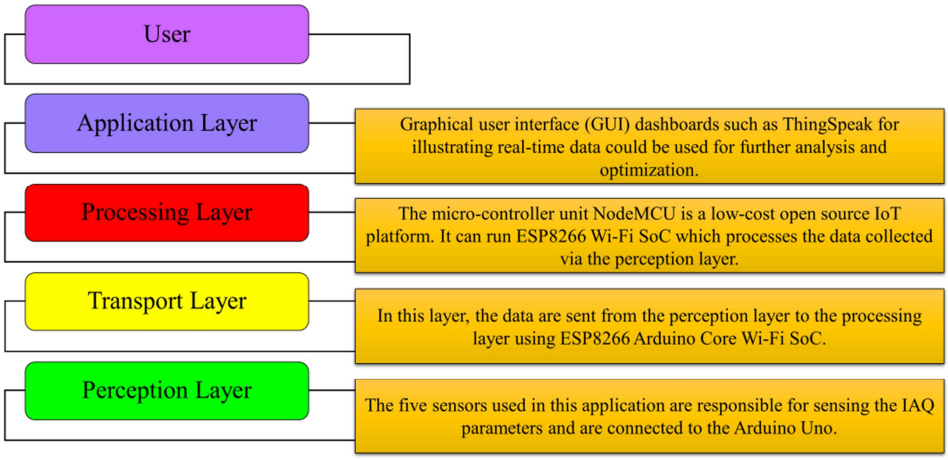


Fig. 7 The layer architecture with implementation details

```
DataTransmitter
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
#include <Wire.h>
#include "Timer.h"
#include "lib/pmsx003/src/pms.h"
#include <SoftwareSerial.h>

//SoftwareSerial ParticleSensor(5, 6);

Pmsx003 pms(D3, D4);

// WiFi parameters
#define WLAN_SSID       "AlRawi Wireless"
#define WLAN_PASS       "xyzxyzxyzxyz"
// Adafruit IO
/*#define AIO_SERVER      "https://thingspeak.com/"
#define AIO_SERVERPORT  1883
#define AIO_USERNAME    "Mohammad"
#define AIO_KEY         "FZ7Y8QL5IBYB0VOL"
*/
// Wifi Object
WiFiClient client;
```

Fig. 8 Wi-Fi parameters for the DataTransmitter program

The proposed device ESP8266 operates as the gateway of the system to the Internet and is a more powerful MCU as it can support more sensors; the modularity of the system enables the interchangeability of sensors. This is beneficial for international applications, where some IEQ parameters are more concerning than others in different countries or regions due to their specific environmental factors.

In this study, the proposed system is also compared with the work of Al Rasyid et al. [6] and Sudarsono et al. [21] in terms of designing a monitoring system through WSN using IoT. Al Rasyid et al. [6] implemented CO and CO2 sensors on a gas sensor motherboard to monitor environmental gas conditions. They sensed the data using MySQL database on meshlium gateway using ZigBee wireless to visualize the environment's condition remotely. Sudarsono et al. [21] also used WSN and added extra sensors to measure T, RH, luminosity, noise, CO, and CO_2. They encrypted the sensors' data and propagated the data through an IEEE802.15.4-based communication gateway for temporary storage. One of the advantages of the system is the flexibility to add extra sensors to measure more IEQ parameters using the ESP8266 MCU. This additional functionality could be programmed using the IDE in Microsoft Visual Studio by the data collector and transmitter. In addition, the current system does not require data storage as the free access to ThingSpeak captures and stores the data. Table 7 shows the present proposed design of hardware and software compared against the work of Al Rasyid et al. [6] and Sudarsono et al. [21]. The present proposed design avoids the use of IEEE 802.15.4 ZigBee, which is a disadvantage of the other systems, so it is easy to carry and access at any time by communication using IoT through ThingSpeak.

Table 7 Comparison of the hardware and software specifications

Reference [6]	Reference [21]	Current study
1. Computer as server	1. Sensor nodes (Node1~Node3)	1. Arduino Uno and ESP8266 NodeMCU V3 as microcontroller and Wi-Fi module
- Central processing unit (CPU): 3.20 GHz (Intel Core i5) - Memory: 4.0 GB RAM	- Microcontroller ATmega1281 14 MHz - Static random access memory (SRAM) 8 KB - Electrically erasable programmable read-only memory (EEPROM) 4 KB - Flash 128 KB - Real-time clock (RTC) 32 KHz - 802.15.4/ZigBee 2.4 GHz	- Arduino Uno Microcontroller: Microchip ATmega328P Digital I/O pins: 14 (of which 6 can provide PWM output) Flash memory: 32 KB Operating voltage: 5 Volts - ESP8266 Wi-Fi module Microcontroller: Tensilica 32-bit RISC CPU Xtensa LX106 Digital I/O pins: 16 Flash memory: 4 MB Operating voltage: 3.3 V
2. Sensor node and gateway	2. Gateway	2. Sensors and gateway (IoT)
- Waspmote PRO 1.2 - Waspmote Gases 2.0 board - CO sensor (TGS2442) - CO_2 sensor (TGS4161) - Xbee S1 module - Meshlium gateway	- Geode Integrated AMD PCS x86 Processor 500 MHz - Cache memory 128 KB - Random access memory (RAM) 256 MB - Disk 8 GB - Linux Debian kernel-2.6.30 - Wi-Fi Atheros AR5213A 802.11b/g 100 mW - 20 dBm - XBee PRO 802.15.4 2.4 GHz 100 mW - Ethernet controller VIA VT6105M (Rhine III) - GNU C Compiler 4.3	- Particle concentration sensor (PMS5003) - Air quality sensor CO, CO_2 (MQ-135) - Temperature and humidity sensor (DHT-11) - Ultraviolet sensor (XC4518) - Sound sensor (XC4438) - ESP8266 (device used to connect to IoT and publish data to the cloud using HTTP protocol)

Table 7 Comparison of the hardware and software specifications (continued)

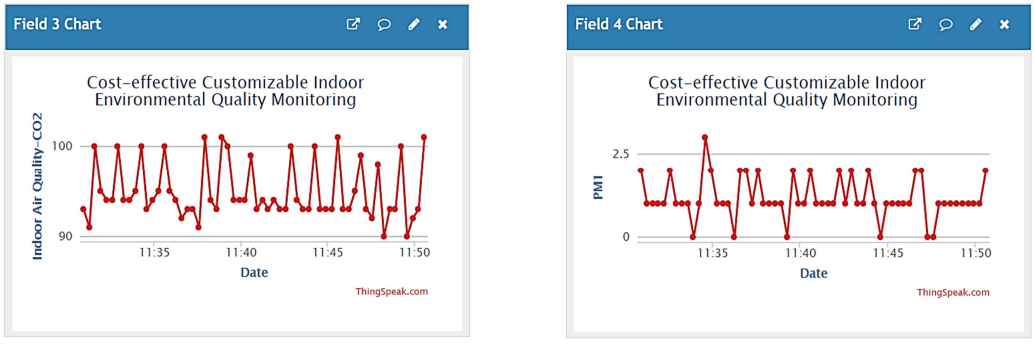
Reference [6]	Reference [21]	Current study
3. Data centre server	3. Data centre server	3. Data centre server
IEEE 802.15.4 ZigBee	- Intel Xeon 3.2 GHz - RAM 4 GB DDR3 - Linux Debian kernel-3.5.0-17 - gcc-4.7.2 - gmp-5.1.1 - pbc-lib-0.5.14 - glib-2.34 - openssl-1.0.1e - Java 1.8.060 - apache-tomcat-8.0.15	- Ryzen 5 processor with 8 GB of RAM - Huawei MateBook D 14 (Ryzen) packs 256 GB of solid-state drive (SSD) storage
4. End user	4. End user	4. End user
- Server computer - Desktop based - Web based	- Intel Core i3 2.4 GHz - RAM 2 GB DDR3 - Wi-Fi 802.11b/g/n - Linux Debian kernel-3.5.0-17 - gcc-4.7.2 - gmp-5.1.1 - pbc-lib-0.5.14 - glib-2.34 - openssl-1.0.1e - Java 1.8.060 - Mozilla Firefox-40.0.3	- Any Android/iOS device that can access/connect to the Internet (Mozilla Firefox and Google Chrome) - Laptop/desktop that can access to the Internet (Mozilla Firefox and Google Chrome)

Also, the link and resource to obtain each item of equipment and software used in this study are provided, and they are easy to be accessed in New Zealand and the Pacific region as shown in Table 8 (which lists the final total cost as NZD $229.84 (USD $162.58)).

Table 8 The total cost for each sensor and board used in the proposed system

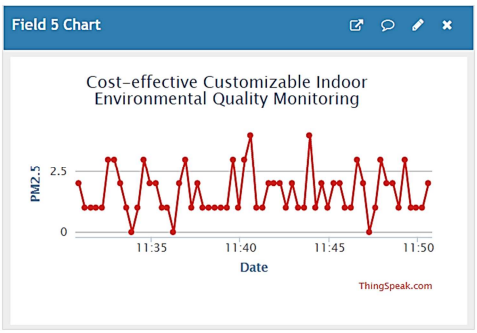
Designator	Equipment	Price
XC-4518	UV sensor	$33.90
XC-4438	Sound sensor	$8.90
PMS5003	Particulate matter sensor	$70.50
MQ-135	Air quality and hazardous gas detection alarm module for Arduino	$5.99
Breadboard	Arduino compatible breadboard with 830 tie points	$19.90
Jumper	Breadboard jumper kit	$13.90
Microcontroller	Arduino Uno	$36.05
DHT-11	Temperature and humidity sensor	$10.70
NodeMCU	ESP8266 v.1	$10.00
Total cost NZD		$209.84

The visualization of the data communicated to the ThingSpeak for CO_2, PM_1, $PM_{2.5}$, VOC, sound level, and UV live readings accessed at different locations from the sensors are shown in Fig. 9. Fig. 9 shows how each parameter is captured in the real time data obtained.

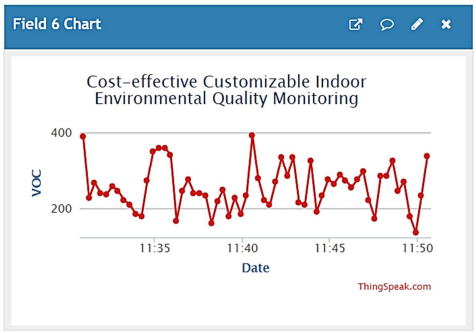


(a) CO_2 (b) PM_1

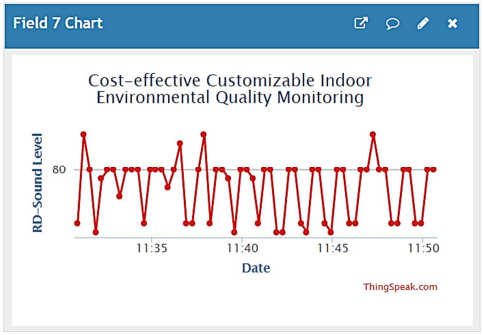
Fig. 9 The dashboard visualization using ThingSpeak live data



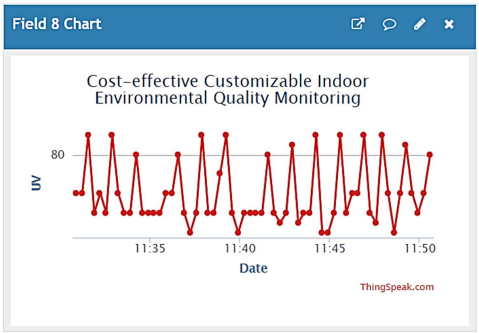
(c) $PM_{2.5}$



(d) VOC



(e) Sound level



(f) UV

Fig. 9 The dashboard visualization using ThingSpeak live data (continued)

Furthermore, the DHT-11 and PMS5003 sensors are validated with an expensive sensor Camfil air image sensor CCSG#01324 manufactured by Camfil AB, Stockholm, Sweden costing NZD860 as shown in Fig. 10. It weighs 200 g with dimensions: W = 144 × H = 64 × D = 61 mm. It operates with 230 V alternating current > 5 V direct current and consumes 10 W. The accuracy of the Camfil sensor for measuring particulate matter ($PM_{2.5}$) is ±0.1 µg/m³ in the operating range 1 to 2.5 µg/m³, for T it is ±0.5ºC with an operating range -10ºC to +50ºC, and for RH (%RH) it is with an accuracy ±2.5% with an operating range 0 to 100 non-condensing [32]. The validation is for three consecutive days, but a 120 minute period is focused on, as shown in Fig. 11 for T, Fig. 12 for RH, and Fig. 13 for particulate matter.

The average temperature measurement error between Camfil sensor and the proposed sensor is 0.55% which is within the range of the accuracy of the sensor. The average RH error is 5.13% and the average error for $PM_{2.5}$ is 3.45%. The testing is conducted in a residential house located in a rural part of Waikato, New Zealand.

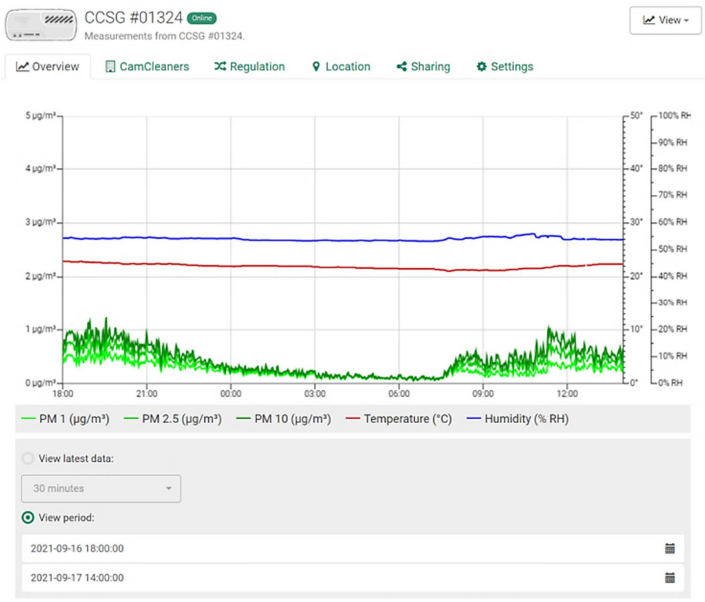


Fig. 10 The air image sensor (Camfil) dashboard

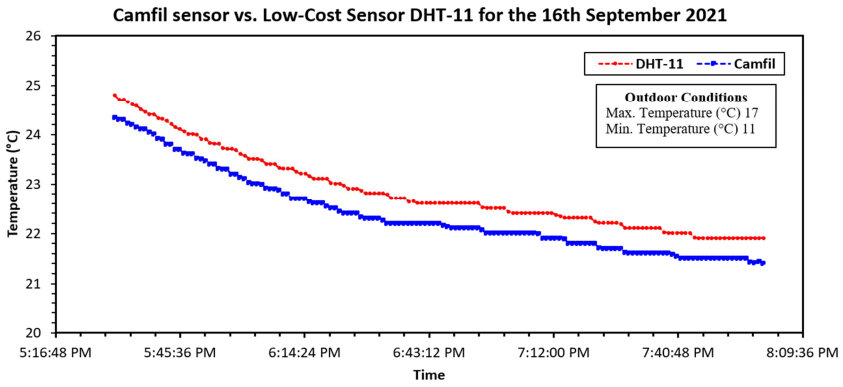


Fig. 11 The DHT-11 temperature validation against the air image sensor (Camfil)

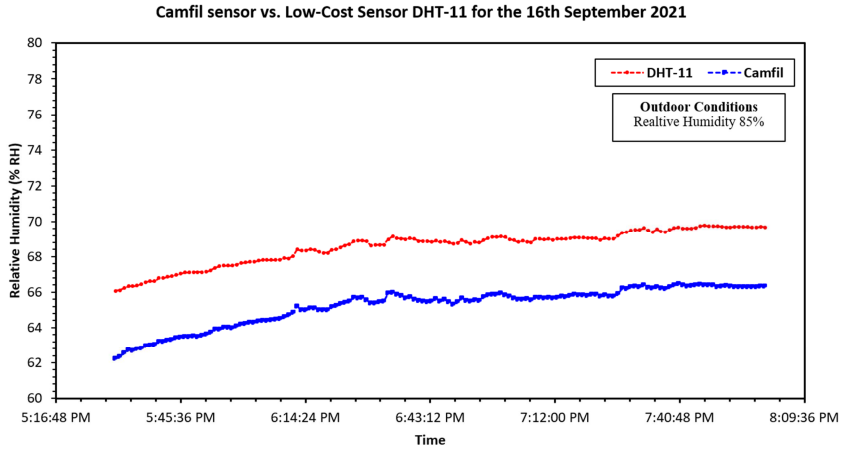


Fig. 12 The DHT-11 relative humidity validation against the air image sensor (Camfil)

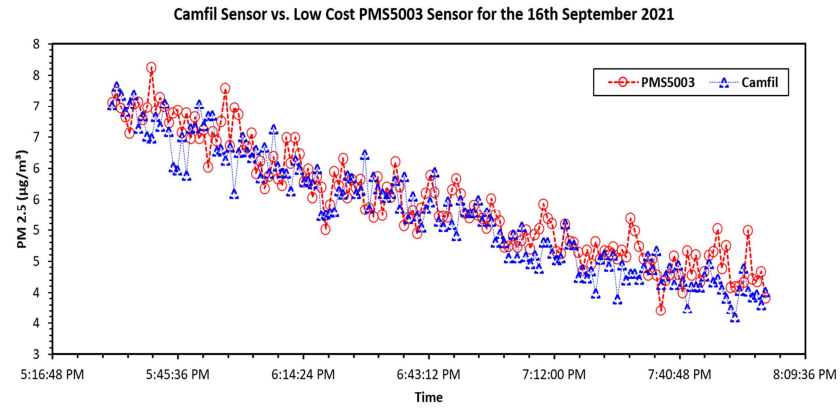


Fig. 13 The PMS5003 particulate matter $PM_{2.5}$ validation against the air image sensor (Camfil)

4. Conclusions

This study presents the steps to assemble and program a low-cost IEQ monitoring system using the IoT comprising five different sensors to monitor the IAQ and IEQ parameters. The current design enables real-time monitoring of the risks and hazard levels of gas emissions, VOCs, and particulate matter, as well as temperature, sound level, and UV concentration for spaces in a residential dwelling. The VOCs, CO_2, and $PM_{2.5}$ real-time readings can be of use to people with respiratory illnesses such as asthma, chronic obstructive pulmonary disease, and allergic disorders. The readings and graphs are easy to obtain with open-source software, and the hardware comprises off-the-shelf low-cost components.

The system presented is validated for these IEQ parameters by benchmarking against the Camfil air image sensor manufactured by Camfil AB, Stockholm, Sweden. The average error of T, RH, and $PM_{2.5}$ are 0.55%, 5.13%, and 3.45%, respectively.

In future work, this system can be tested in use cases, such as with the subjects with respiratory conditions to validate the effectiveness in assessing and improving the IEQ and enhance overall well-being. Also, this could be used in a variety of settings to correlate IEQ parameters with illness amongst the occupants, e.g., office spaces, childcare facilities, or doctor's clinics.

Acknowledgements

The authors would like to thank Wintec Research Office and Centre for Engineering and Industrial Design for internal funding to support this research. The authors wish to acknowledge the support of Jacob Bakker, Stefan Von Maltitz, and Timothy Fargher for helping with the code.

Conflicts of Interest

The authors declare no conflict of interest.

References

[1] P. Markowicz and L. Larsson, "Influence of Relative Humidity on VOC Concentrations in Indoor Air," Environmental Science and Pollution Research, vol. 22, no. 8, pp. 5772-5779, October 2014.
[2] D. Mulenga and S. Siziya, "Indoor Air Pollution Related Respiratory Ill Health, a Sequel of Biomass Use," SciMedicine Journal, vol. 1, no. 1, pp. 30-37, February 2019.
[3] S. Colclough, O. Kinnane, N. Hewitt, and P. Griffiths, "Investigation of nZEB Social Housing Built to the Passive House Standard," Energy and Buildings, vol. 179, pp. 344-359, November 2018.
[4] World Health Organization, "Household Air Pollution and Health," http://www.who.int/news-room/fact-sheets/detail/household-air-pollution-and-health, May 08, 2018.
[5] P. Bluyssen, The Indoor Environment Handbook: How to Make Buildings Healthy and Comfortable, 1st ed., London: Routledge, 2009.
[6] M. U. H. Al Rasyid, I. U. Nadhori, A. Sudarsono, and Y. T. Alnovinda, "Pollution Monitoring System Using Gas Sensor Based on Wireless Sensor Network," International Journal of Engineering and Technology Innovation, vol. 6, no. 1, pp. 79-91, January 2016.
[7] J. G. Allen and J. D. Macomber, "We Spend 90% of Our Time Inside—Why Don't We Care That Indoor Air Is So Polluted?" https://www.fastcompany.com/90506856/we-spend-90-of-our-time-inside-why-dont-we-care-that-indoor-air-is-so-polluted, May 20, 2020.
[8] J. C. Nwanaji-Enwerem, J. G. Allen, and P. I. Beamer, "Another Invisible Enemy Indoors: COVID-19, Human Health, the Home, and United States Indoor Air Policy," Journal of Exposure Science and Environmental Epidemiology, vol. 30, no. 5, pp. 773-775, July 2020.
[9] L. Morawska, J. W. Tang, W. Bahnfleth, P. M. Bluyssen, A. Boerstra, G. Buonanno, et al., "How Can Airborne Transmission of COVID-19 Indoors be Minimised?" Environment International, vol. 142, 105832, September 2020.
[10] J. McCormack, "Coronavirus: NI to Face New Lockdown Measures from Next Friday," https://www.bbc.com/news/uk-northern-ireland-55004210, November 19, 2020.
[11] R. Mumtaz, S. M. H. Zaidi, M. Z. Shakir, U. Shafi, M. M. Malik, A. Haque, et al., "Internet of Things (IoT) Based Indoor Air Quality Sensing and Predictive Analytic—A COVID-19 Perspective," Electronics, vol. 10, no. 2, 184, January 2021.
[12] H. Zhang, R. Srinivasan, and V. Ganesan, "Low Cost, Multi-Pollutant Sensing System Using Raspberry Pi for Indoor Air Quality Monitoring," Sustainability, vol. 13, no. 1, 370, January 2021.
[13] A. Baldelli, "Evaluation of a Low-Cost Multi-Channel Monitor for Indoor Air Quality through a Novel, Low-Cost, and Reproducible Platform," Measurement: Sensors, vol. 17, 100059, October 2021.
[14] A. Baldelli, M. Jeronimo, M. Tinney, and K. Bartlett, "Real-Time Measurements of Formaldehyde Emissions in a Gross Anatomy Laboratory," SN Applied Sciences, vol. 2, no. 4, 769, April 2020.
[15] P. Kheirkhah, A. Baldelli, P. Kirchen, and S. Rogak, "Development and Validation of a Multi-Angle Light Scattering Method for Fast Engine Soot Mass and Size Measurements," Aerosol Science and Technology, vol. 54, no. 9, pp. 1083-1101, May 2020.
[16] A. Baldelli, M. Jeronimo, B. Loosley, G. Owen, I. Welch, and K. Bartlett, "Particle Matter, Volatile Organic Compounds, and Occupational Allergens: Correlation and Sources in Laboratory Animal Facilities," SN Applied Sciences, vol. 2, no. 10, 1672, October 2020.
[17] A. Baldelli, J. Ou, W. Li, and A. Amirfazli, "Spray-On Nanocomposite Coatings: Wettability and Conductivity," Langmuir, vol. 36, no. 39, pp. 11393-11410, August 2020.
[18] T. Parkinson, A. Parkinson, and R. de Dear, "Continuous IEQ Monitoring System: Context and Development", Building and Environment, vol. 149, pp. 15-25, February 2019.

[19] J. Jo, B. Jo, J. Kim, S. Kim, and W. Han, "Development of an IoT-Based Indoor Air Quality Monitoring Platform," Journal of Sensors, vol. 2020, 8749764, January 2020.
[20] G. Marques and R. Pitarma, "A Cost-Effective Air Quality Supervision Solution for Enhanced Living Environments through the Internet of Things," Electronics, vol. 8, no. 2, 170, February 2019.
[21] A. Sudarsono, S. Huda, N. Fahmi, M. U. H. Al-Rasyid, and P. Kristalina, "Secure Data Exchange in Environmental Health Monitoring System through Wireless Sensor Network," International Journal of Engineering and Technology Innovation vol. 6, no. 2, pp. 103-122, April 2016.
[22] M. Taştan and H. Gökozan, "Real-Time Monitoring of Indoor Air Quality with Internet of Things-Based E-Nose," Applied Sciences, vol. 9, no. 16, 3435, August 2019.
[23] D. Wall, P. McCullagh, I. Cleland, and R. Bond, "Development of an Internet of Things Solution to Monitor and Analyse Indoor Air Quality," Internet of Things, vol. 14, 100392, June 2021.
[24] N. Dinh and S. Lim, "Performance Evaluations for IEEE 802.15.4-Based IoT Smart Home Solution," International Journal of Engineering and Technology Innovation, vol. 6, no. 4, pp. 274-283, September 2016.
[25] G. Parmar, S. Lakhani, and M. K. Chattopadhyay, "An IoT Based Low Cost Air Pollution Monitoring System," International Conference on Recent Innovations in Signal processing and Embedded Systems, pp. 524-528, October 2017.
[26] M. Elsisi, K. Mahmoud, M. Lehtonen, and M. M. Darwish, "Reliable Industry 4.0 Based on Machine Learning and IoT for Analyzing, Monitoring, and Securing Smart Meters," Sensors, vol. 21, no. 2, 487, January 2021.
[27] M. Elsisi, M. Q. Tran, K. Mahmoud, M. Lehtonen, and M. M. Darwish, "Deep Learning-Based Industry 4.0 and Internet of Things towards Effective Energy Management for Smart Buildings," Sensors, vol. 21, no. 4, 1038, February 2021.
[28] M. Q. Tran, M. Elsisi, K. Mahmoud, M. K. Liu, M. Lehtonen, and M. M. Darwish, "Experimental Setup for Online Fault Diagnosis of Induction Machines via Promising IoT and Machine Learning: Towards Industry 4.0 Empowerment," IEEE Access, vol. 9, pp. 115429-115441, August 2021.
[29] M. Elsisi, M. Q. Tran, K. Mahmoud, D. E. A. Mansour, M. Lehtonen, and M. M. Darwish, "Towards Secured Online Monitoring for Digitalized GIS Against Cyber-Attacks Based on IoT and Machine Learning," IEEE Access, vol. 9, pp. 78425-78427, May 2021.
[30] The New Zealand Government, "Resource Management (National Environmental Standards for Air Quality) Regulations 2004," https://www.legislation.govt.nz/regulation/public/2004/0309/latest/DLM286835.html?search=ta_regulation_R_rc%40rinf%%40rnif_an%40bn%40rn_25_a&p=3, September 01. 2020.
[31] H. P. Halvorsen, "ThingSpeak," https://www.halvorsen.blog/documents/technology/iot/thingspeak/thingspeak.php, 2021.
[32] M. Al-Rawi, C. A. Ikutegbe, A. Auckaili, and M. M. Farid, "Sustainable Technologies to Improve Indoor Air Quality in a Residential House—A Case Study in Waikato, New Zealand," Energy and Buildings, vol. 250, 111283, July 2021.

Appendix A

DataCollector program

```
#include <Wire.h>
#include "lib/DHT_sensor_library/DHT.h"
#include "Timer.h"

// DHT Defines
#define DHTPIN 2
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);

#define DHpin 8
byte dat[5];

Timer I2CTimer;
Timer AnalogReadTimer;
Timer TempratureTimer;

int UVReading = 0;
int highestSoundLevel = 0;
int AirQuality = 0;
float humidity = 0.0f;
float temprature = 0.0f;

void setup()
{
    // put your setup code here, to run once:
    Serial.begin(115200);
```

```
    delay(100);
    dht.begin();
    // I2C Init and Listen Function
    Wire.begin(20);
    Wire.onRequest(DataRequested);

    I2CTimer.Reset();
    AnalogReadTimer.Reset();
    Serial.println("init complete");

    pinMode(DHpin, OUTPUT);
}


void loop()
{
    // put your main code here, to run repeatedly:
   // I2CRead();
    ReadAnalogSignals();
    ReadTempAndHumidity();


   // delay(700);
}

// Reads in analog Signals and stores them in the global variables
void ReadAnalogSignals()
{
    if (AnalogReadTimer.TimePassed(100, true))
    {
        // Set analog refernce voltage to the INTERNAL 1.1V Ref
        analogReference(INTERNAL);
        analogRead(A0);
        delay(2);
        // store analog reading in the UVReading Variable
        UVReading = analogRead(A0);
      //  Serial.print("UVReading: "); Serial.println(UVReading);
        // Reset to DEFAULT Value (5V Ref)
        analogReference(DEFAULT);
        analogRead(A0);
        delay(2);

       // Serial.print("air qual:");
        //Serial.println(AirQuality);
    }
    // create a tempoary soundIn var to store the sound reading.
    int soundIn = analogRead(A1);

    // updates the highestSoundLevel if the soundIn reading is higher than the current recorded value
    if (soundIn > highestSoundLevel)
    {
        highestSoundLevel = soundIn;
      //  Serial.print("Sound Peak: "); Serial.println(highestSoundLevel);
    }

    AirQuality = analogRead(A2);

    double testdouble = 23.23;
}

// sends data back to ESP8266  using I2C (Wire.h)
void DataRequested()
{
    Serial.println("data req");
    uint8_t sendArray[14];
    sendArray[0] = highestSoundLevel >> 8;
    sendArray[1] = highestSoundLevel;
    sendArray[2] = UVReading >> 8;
    sendArray[3] = UVReading;
    sendArray[4] = AirQuality >> 8;
    sendArray[5] = AirQuality;

    uint8_t tempArray[4];
    memcpy(tempArray, &temprature, 4);
    sendArray[6] = tempArray[0];
    sendArray[7] = tempArray[1];
    sendArray[8] = tempArray[2];
    sendArray[9] = tempArray[3];
```

```
    memcpy(tempArray, &humidity, 4);
    sendArray[10] = tempArray[0];
    sendArray[11] = tempArray[1];
    sendArray[12] = tempArray[2];
    sendArray[13] = tempArray[3];

    Serial.print("data::");
    Serial.print(tempArray[0]);
    Serial.print(tempArray[1]);
    Serial.print(tempArray[2]);
    Serial.println(tempArray[3]);

    Wire.write(sendArray,14);
  //  Serial.println(temp[0]);
    //Serial.println(temp[1]);
    highestSoundLevel = 0;
}

void ReadTempAndHumidity()
{
    if (TempratureTimer.TimePassed(1000, true))
    {
        // Reading temperature or humidity takes about 250 milliseconds!
        // Sensor readings may also be up to 2 seconds 'old' (its a very slow sensor)
        humidity = dht.readHumidity();
        // Read temperature as Celsius (the default)
        temprature = dht.readTemperature();

        // Check if any reads failed and exit early (to try again).
        if (isnan(humidity) || isnan(temprature))
        {
            Serial.println(F("Failed to read from DHT sensor!"));
            humidity = 0;
            temprature = 0;
            return;
        }

        // Compute heat index in Celsius (isFahreheit = false)
        float temprature = dht.computeHeatIndex(temprature, humidity, false);

        Serial.print(F("Humidity: "));
        Serial.println(humidity);
        Serial.print("Temprature: ");
        Serial.print(temprature);
        Serial.print(F("°C "));
    }
}
```

Appendix B

DataTransmitter

```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
#include <Wire.h>
#include "Timer.h"
#include "lib/pmsx003/src/pms.h"
#include <SoftwareSerial.h>

//SoftwareSerial ParticleSensor(5, 6);

Pmsx003 pms(D3, D4);

// WiFi parameters
#define WLAN_SSID       "AlRawi Wireless"
#define WLAN_PASS       "xyzxyzxyzxyz"
// Adafruit IO
/*#define AIO_SERVER      "https://thingspeak.com/"
#define AIO_SERVERPORT  1883
#define AIO_USERNAME    "Mohammad"
#define AIO_KEY         "FZ7Y8QL5IBYB0VOL"
*/
// Wifi Object
WiFiClient client;

// HTTP Object
HTTPClient http;
```

```
//I2C Buffer Vars
uint8_t inCount = 0;
uint8_t inData[30];

// Retrived Data Values
int RD_SoundLevel = 0;
int RD_UVLevel = 0;
int RD_AirQuality = 0;
float RD_Humidity = 0;
float RD_Temprature = 0;
int RD_pm1 = 0;
int RD_pm2 = 0;
int RD_pm10 = 0;
//Timers
Timer FetchDataTimer;
Timer ThingSpeakUpdateTimer;

Timer TestTimer;

void setup()
{
    Serial.begin(115200);
    Serial.println(F("Adafruit IO Example"));
    // Connect to WiFi access point.
    Serial.println(); Serial.println();
    delay(10);
    Serial.print(F("Connecting to "));
    Serial.println(WLAN_SSID);
    WiFi.begin(WLAN_SSID, WLAN_PASS);

    Wire.begin();

    while (WiFi.status() != WL_CONNECTED)
    {
        delay(500);
        Serial.print(F("."));
    }
    Serial.println();
    Serial.println(F("WiFi connected"));
    Serial.println(F("IP address: "));
    Serial.println(WiFi.localIP());

  pms.begin();
  pms.waitForData(Pmsx003::wakeupTime);
  pms.write(Pmsx003::cmdModeActive);


    FetchDataTimer.Reset();
  ThingSpeakUpdateTimer.Reset();
}

void loop()
{
    DataRequest();
  ReadParticalSensor();

  if (ThingSpeakUpdateTimer.TimePassed(20000, true))
  {
    SendHTTPData();
  }
}


auto lastRead = millis();

void ReadParticalSensor()
{
  const auto n = Pmsx003::Reserved;
  Pmsx003::pmsData data[n];

  Pmsx003::PmsStatus status = pms.read(data, n);

  switch (status) {
  case Pmsx003::OK:
  {
    Serial.println("____________________");
    auto newRead = millis();
    Serial.print("Wait time ");
```

```
    Serial.println(newRead - lastRead);
    lastRead = newRead;

    // For loop starts from 3
    // Skip the first three data (PM1dot0CF1, PM2dot5CF1, PM10CF1)
    for (size_t i = Pmsx003::PM1dot0; i < n; ++i) {
      Serial.print(data[i]);
      Serial.print("\t");
      Serial.print(Pmsx003::dataNames[i]);
      Serial.print(" [");
      Serial.print(Pmsx003::metrics[i]);
      Serial.print("]");
      Serial.println();
    }
    RD_pm1 = data[4];
    RD_pm2 = data[5];
    RD_pm10 = data[6];

    break;
  }
  case Pmsx003::noData:
    //Serial.println("noData");
    break;
  default:
    Serial.println("_________________");
    Serial.println(Pmsx003::errorMsg[status]);
  };
}

void DataRequest()
{
    if (FetchDataTimer.TimePassed(1000, true))
    {
        Serial.println("Asking for Data");
        Wire.requestFrom(20, 14);
    }


  if (Wire.available())
  {
    uint8_t inbyte = Wire.read();
    Serial.println(inbyte);
    inData[inCount] = inbyte;
    inCount++;
  }

  if (inCount == 14)
  {
    Serial.println("I2C Data Updated");
    RD_SoundLevel = inData[0] << 8;
    RD_SoundLevel += inData[1];
    RD_UVLevel = inData[2] << 8;
    RD_UVLevel += inData[3];
    RD_AirQuality = inData[4] << 8;
    RD_AirQuality += inData[5];
    //+= (equivalent) = RD_AirQuality = RD_AirQuality + inData[5];

    // Rebuild Float data from raw byte data
    *((uint8_t*)(&RD_Temprature) + 3) = inData[9];
    *((uint8_t*)(&RD_Temprature) + 2) = inData[8];
    *((uint8_t*)(&RD_Temprature) + 1) = inData[7];
    *((uint8_t*)(&RD_Temprature) + 0) = inData[6];
    Serial.print("temp: "); Serial.println(RD_Temprature);

    // Rebuild Float data from raw byte data
    *((uint8_t*)(&RD_Humidity) + 3) = inData[13];
    *((uint8_t*)(&RD_Humidity) + 2) = inData[12];
    *((uint8_t*)(&RD_Humidity) + 1) = inData[11];
    *((uint8_t*)(&RD_Humidity) + 0) = inData[10];
    Serial.print("humidity: "); Serial.println(RD_Humidity);

    Serial.print("Sound level set to: "); Serial.println(RD_SoundLevel);
    inCount = 0;
  }
}

// Packages data and Sends to Thingspeak Rest
void SendHTTPData()
{
```

```
  //https://api.thingspeak.com/update?api_key=FZ7Y8QL5IBYB0VOL&field1=0
  String httpUrl = "http://api.thingspeak.com/update?api_key=FZ7Y8QL5IBYB0VOL";

  String dataString = "&field7=" + (String)RD_SoundLevel
    + "&field8=" + (String)RD_UVLevel
    + "&field3=" + (String)RD_AirQuality
    + "&field1=" + (String)RD_Temprature
    + "&field2=" + (String)RD_Humidity
    + "&field4=" + (String)RD_pm1
    + "&field5=" + (String)RD_pm2
    + "&field6=" + (String)RD_pm10;


  httpUrl += dataString;

  Serial.println(httpUrl);
  String payload = "";
  if (http.begin(client, httpUrl));
  {
    int httpCode = http.GET();

    // httpCode will be negative on error
    if (httpCode > 0)
    {
      if (httpCode == HTTP_CODE_OK)
      {
        payload = http.getString();
        Serial.println(payload);
      }
    }
    else
    {
      Serial.print(F("[HTTP] GET... failed, error: )"));
      payload = http.errorToString(httpCode).c_str();
      Serial.println(payload);
    }

    Serial.print("http code: "); Serial.println(httpCode);

    http.end();
    client.flush();
    client.stop();
  }

  //return payload;
}
```