A CO_2 monitor as an introductory microelectronics project helping to slow-down the spread of the corona virus and ensuring a healthy learning and working environment

A. Köhn-Seemann, alf.koehn@gmail.com

2021-01-25

Abstract

This paper describes the setup of a simple yet reliable CO_2 monitor which is based on open-source microelectronics hardware. The monitor is intended to be used in class rooms, lecture halls or offices and can be constructed as a joint students project. It was motivated by recent discussions on the role of aerosols being part of exhaled air to spread the corona virus. The aerosol concentration in air is correlated with the CO_2 concentration. Measuring the latter can thus help to slow-down the spread of the corona-virus. The program code used for the CO_2 monitor and this documentation is available as a GitHub repository to allow for updates and improvements.

First full version released on 2020-09-22.

1 Introduction

It is generally accepted that the CO_2 concentration in a class room has an influence on students' activities, their ability to study and learn [1, 2], or on their health and thus attendance [3]. The same applies of course to office environments [4]. The major source of CO_2 in a class room is the exhaled air of the students (and teachers) [5]. It thus increases over time but can also be relatively easy controlled by proper ventilation. Monitoring the CO_2 concentration over time provides thus a simple way to ensure a productive and healthy learning environment.

In addition to CO_2 , exhaled air consists of aerosols (among other things). It is now generally accepted that the aerosols of patients being infected with SARS-CoV-2, might contain viable virus concentrations which are large enough to cause further infections if somebody else inhales those aerosols [6–9]. Note that this seems to happen even if the infected patients show no symptoms of SARS-CoV-2 [10] or not yet any symptoms [11]. It is thus not surprising that the vast majority of SARS-CoV-2 virus transmissions seems to happen indoors [12] and that closing schools and universities, for example, were found to have a significant effect on the spread of the virus [13]. Face masks can help to reduce the aerosol outflow and thus also slow down the infection rates [14, 15].

With half-life periods of the virus on aerosols on the order of 1 hour [16], it becomes evident that proper ventilation, strongly reducing the aerosol concentration, can help to prevent hidden infections, i.e. infections where the infected person is not (yet) aware of their infection but already contagious. This is further stressed by a case study from a seafood market in south China [17], a restaurant also from China [18], or from a choir in the US [19]. Note that ventilation was already recommended as a proper measure against the spread of a different pandemic, the Spanish flu, more than 100 years ago [20].

Since aerosols and CO_2 are both parts of exhaled air, measuring the CO_2 concentration in a room provides an easy accessible indicator for the aerosol concentration [21, 22]. In recent recommendations from national authorities, it was suggested to use the CO_2 concentration as an indicator when ventilation is required [23–25]. A relevant example for the positive effect of proper ventilation based on the CO_2 concentration in a room is the stopping of a tuberculosis outbreak at the Taipei University in Taiwan: only after the air circulation in every room was improved such that the CO_2 concentration stayed around 600 ppm (the outdoor value is approximately 400 ppm), the outbreak came to a halt and stopped completely [26].

While commercial CO_2 monitors do exist [27–29], these might be considered too expensive for usage in large quantities in schools or universities and/or currently have long delivery times (since their potential help in slowing down the spread of the SARS-CoV-2 virus seems to become more and more accepted). Here we present a simple and cost effective, yet reliable way to monitor the CO_2 concentration. Widely available microelectronic components are used which can be easily programmed via open source software platforms allowing to modify and extend the examples presented in this paper. Students can build the detectors in class as a joint project which might serve to raise interest in electronics or the underlying physical and chemical processes [30].

This work was inspired by a project of the *Hochschule Trier* [31], where the design and construction of a CO_2 measuring device is suggested as a students' project, allowing to discuss a variety of scientific topics during the course of the project. In addition, a few posts from different forums served as an inspiration [32–35]. Furthermore, a small number of projects hosted and maintained as GitHub repositories and webpages using the same CO_2 detector are available [36–39]. We would like to recommend the interested reader in particular to the repository by paulvha [38] as it contains a rather large number of examples and to an article published in the *Make Magazin* [40] which contains a full description including the assembly of a CO_2 monitor similar to the one presented in this paper.

2 The CO_2 monitor

The CO_2 monitor is based on the microelectronic sensor SCD30 which measures the CO_2 concentration and also provides measurements of the ambient temperature and relative humidity [41]. Using Arduino as a programing language and some microcontroller, it is straightforward to get the sensor running and outputting data, thanks to the examples available in the libraries provided by SparkFun [42]. Using the Arduino IDE [43], which is available for all major operating systems, the corresponding libraries can be simply included via the library manager.

To make the CO_2 monitor visually appealing, we decided to output the measurement to an OLED display (which is inexpensive and available in a large variety of sizes and configurations). Due to the widespread usage of such displays, they can also be directly included via the library manager in the Arduino IDE. Alternatively, an LCD display can be used which has the advantage of being larger in diameter but having a reduced resolution at a similar price. Note that we recommend to use LCD-displays with I2C-modules to make the wiring and/or soldering less complicated (connecting 4 cables instead of 16).

In addition to just showing some numbers, we have included a red LED at the prototype installation which lights up as soon as some threshold value of the CO_2 concentration is reached, indicating the need for ventilation. One could also think of a traffic light design, where first a yellow LED lights up at a slightly lower threshold value. The *Federation of European Heating, Ventilation and Air Conditioning associations (REHVA)* recommends to issue a warning, corresponding to an orange light, when a value of 800 ppm is

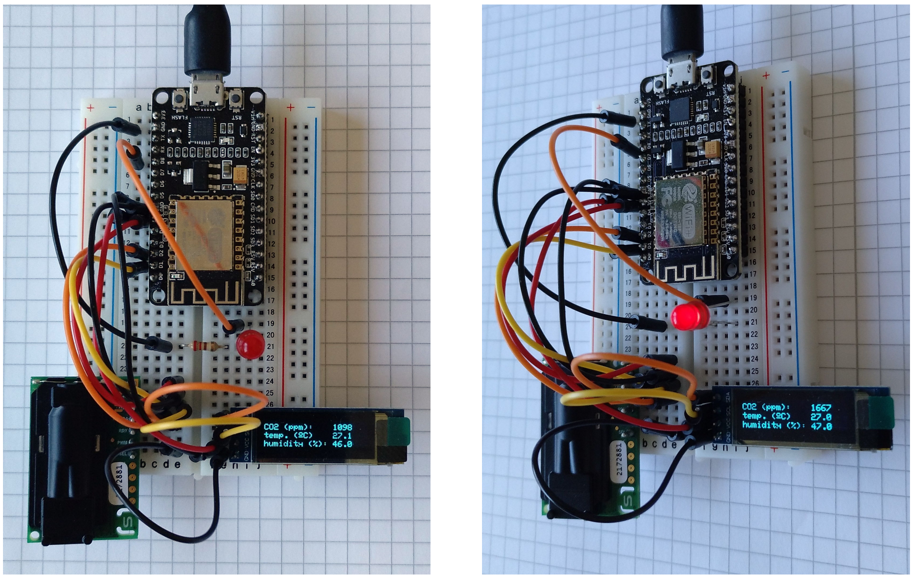

Figure 1: Assembled and working prototype of the CO_2 monitor, *(left)* with a measured CO_2 concentration below the threshold and *(right)* above it (note the red LED).

reached and prompt to trigger some action like ventilation, corresponding to a red light, when 1000 ppm are reached [44]. The *Federal Ministry of Labour and Social Affairs* of Germany also states a threshold value of 1000 ppm that should not be passed [45]. Note that a value of approximately 410 ppm is the typical CO_2 concentration of fresh air [46].

As a microcontroller we decided to use the low-cost open source NodeMCU ESP8266 board [47], as it offers enough flexibility to further extend the functionality of the CO_2 monitor. Of particular interest might be the WiFi capability allowing for example to write the measured values to a web-server where they can then be accessed via a web-browser or an app on a smartphone.

A prototype of the CO_2 monitor is shown in Fig. 1. As one can see, it is not enclosed in some box to still allow easy access for modifications. The idea of this prototype was rather to show that the general principle of the CO_2 monitor is working and not to provide a polished final product. The prototype is ready to be used in a class room or lecture hall, although it might be worth to mount everything into a box which is not only visually more appealing but provides also some protection.

As a next step one might want to replace the breadboard by a stripboard which of course requires some soldering but results in a more robust device.

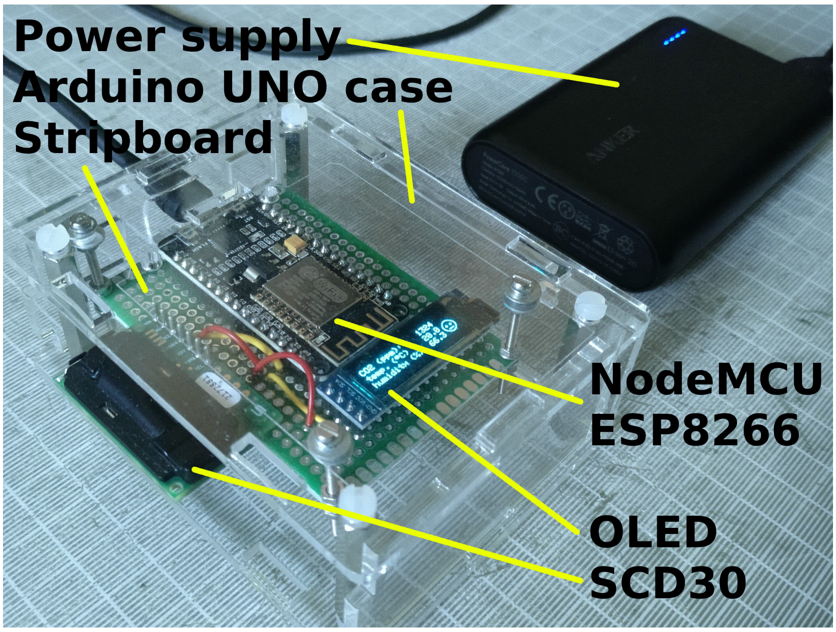

Figure 2: Assembled CO_2 monitor, version 2. Note the usage of a stripboard and a casing originally designed for an Arduino Uno. In addition to the actual value, the CO_2 concentration is indicated by the happiness of a smiley face shown next to the numbers on the OLED display.

Instead of designing a case for the CO_2 monitor, it is possible to use cases which were originally designed for a Raspberry Pi or an Arduino Uno. Furthermore, the LEDs as an indicator for the level of the CO_2 concentration can be replaced by a smiley face on the OLED display whose happiness correlates with the CO_2 concentration: a higher level results in a more sad face. Figure 2 shows a photography of such an assembly, referred to as CO_2 monitor version 2. More details about different assembly variations are discussed in Section 5.

2.1 Positioning in a room

The CO_2 monitor should be placed at a position in a room where it is not exposed to flowing air as this would distort the measured CO_2 concentration (see Section 4). This means that it should, for example, not be placed in-between a window and a door if both are used for efficient cross ventilation[1].

[1] Altough cross ventilation is more efficient than impact ventilation ("querlüften" vs. "stoßlüften" [48]) one has to take care that behind the open door in Fig. 3 another open window is needed otherwise one could release part of the classroom air and its potentially

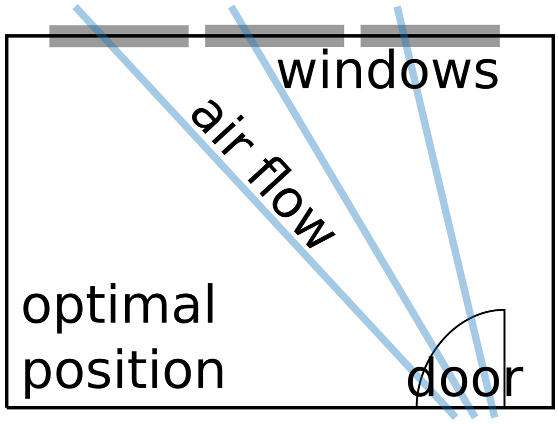

Figure 3: Optimal position of the CO_2 monitor in a room where ventilation by an air flow across the room (cross ventilation) can be applied.

Figure 3 indicates the optimal positioning in such a situation. It should also not be positioned too close to the students' or teachers' heads as they could temporarily trigger very high values being displayed on the CO_2 monitor if directly exhaling onto it. A position slightly above everybody's head seems to be best as this would also allow everybody to have a look at it. The latter fact could in principle lead to some students' closely following every change on the CO_2 monitor instead of paying attention to the class or lecture. An easy solution would be to show only CO_2 concentrations rounded to 100 ppm, for example print "`CO₂ in 100 ppm:  8`" to a display [49].

Also be aware that classrooms and lecture halls often provide only a small number of wall sockets. One would thus need either a long enough power cable or a power bank (USB charger) which can also be included in the case of the CO_2 monitor itself.

3 Required parts

The CO_2 monitor as presented here consists of a number of parts for which it is not important to use the exact same model. The only component which should not be replaced is the CO_2 measuring device, the SCD30. Note that the program code discussed in Sec. 6 is tailored for the NodeMCU ESP8266, replacing that component would thus require small adjustments to the code.

The parts used for the prototype of the CO_2 monitor are listed in Table 3. The display can be easily replaced by an OLED of larger size. One could also

infectious aerosols into the corridor which connects different classrooms or lecture halls [49].

Element	Quantity	Price
SCD30 (CO_2 sensor)	1	45 €
NodeMCU ESP8266	1	5 €
0.91″ OLED display	1	5 €
red LED	1	0.2 €
220 Ω resistor	1	0.1 €
mini breadboard	1	4 €
breadboard cables	10	4 €
pin header	1	0.5 €
micro USB cable	1	3 €

Table 1: Components used for the CO_2 monitor as presented in this paper (note that the prices were obtained in 09/2020 and may vary).

use multiple displays, which would require to take care of proper addressing the displays and thus add a little bit of complexity to the code (and to the assembly).

The usage of a breadboard was motivated by educational purposes as this allows very easy assembly without the need to solder anything. It can, however, directly be replaced by a stripboard or completely omitted and use only cables or pin headers (which would require some soldering).

Note that the prices as listed in the table can be pushed down (significantly for some of the components) when ordering larger quantities.

For the prototype design of the CO_2 monitor we have decided to leave out a proper casing. One could either use a standard-sized case, or design one and print it for example on a 3D printer or re-use/recycle some old boxes. It is however important to correctly position the SCD30 inside the box: as described in a manufacturer's document [50], the sensor is ideally placed as close as possible to the box's outer shell and to a large opening to be properly exposed to the ambient. The box should be as small as possible to get fast response times to changes in the ambient air. The SCD30 should also be isolated from direct air flow, as the corresponding changes in pressure (due to the air flow) would lead to increased noise and thus reduced accuracy in the measurements. It is also recommended to not directly place the sensor above heat sources like for example microcontrollers.

4 The CO_2 sensor

The SCD30 has been chosen because it performs direct measurements of the CO_2 concentration. Cheaper sensors often measure the concentration of volatile organic compounds (VOC) and then assume a correlation between the two quantities. This can, however, lead to wrong values of the CO_2 concentration since VOC can be emitted from a variety of chemicals. Although VOCs are also known to cause health problems, here we are explicitly interested in the CO_2 concentration, as discussed in Sec. 1. For a discussion about monitoring VOC and CO_2 concentration with self-assembled devices we would like to point the interested reader to e.g. Ref. [51].

4.1 Technical specifications

According to the datasheet of the SCD30 [41], the CO_2 sensor has a measurement range of $0 - 40,000$ ppm with an accuracy of ± 30 ppm. The supply voltage needs to be between 3.3 and 5 V which allows to use a variety of microcontrollers. The drawn current is specified to be on average 19 mA with a maximum value of 75 mA. With a sensor lifetime of 15 years, the SCD30 offers a reliable system to permanently monitor the CO_2 concentration.

4.2 Nondispersive infrared technique

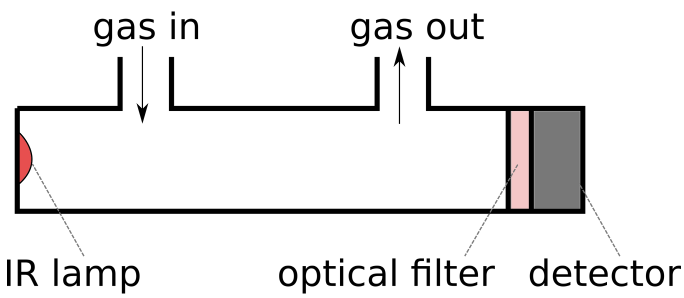

Figure 4: Sketch of a sensor using the nondispersive infrared technique to measure CO_2 concentration.

The CO_2 concentration is measured using the so-called *nondispersive infrared* technique (NDIR) which is the most common sensor type used in industry to measure the CO_2 concentration [52–54]. Its principle is sketched in Fig. 4. A light source (a light bulb is used here) emits infrared light which travels through a tube filled with a sample of the surrounding air. The spectrum of the emitted light includes the $4.26\,\mu$m absorption band of CO_2 which is unique to the typical components of air and the light is absorbed by them. At the end of the tube, the remaining light hits an optical filter that allows only that specific wavelength of $4.26\,\mu$m to pass. A CMOS detector then collects the remaining light and measures its intensity I_1. The difference between the intensity of light emitted by the source I_0 and received by the detector at this specific wavelength is due to the CO_2 molecules in the tube which then allows to calculate the CO_2 concentration using the Beer-Lambert law:

$$I_1 = I_0 e^{-\kappa C l}, \tag{1}$$

where κ is the absorption coefficient of CO_2 , C its concentration, and l the length of the tube. A second tube without the optical filter in-front of the CMOS detector is used as a reference tube to compensate variations of I_0. Using folded optics, i.e. waveguides, for the tube allows for a very compact size of the overall sensor on the order of just a few centimeters.

4.3 Calibration

The SCD30 is sold as a fully calibrated sensor and thus requires in principle no calibration before its usage. According to the manual [55], the sensor is set to automatically perform a self-calibration. This requires, however, to expose the sensor to fresh air on a regular basis. In particular during the first 7 days of operation, it has to be exposed to fresh air for one hour every day [55]. Since this is a requirement which is unrealistic to fulfill for our use case, we decided to follow a different approach: instead of the *automatic self-calibration (ASC)*, a *forced recalibration (FRC)* can be performed after triggering it by the user. According to our observations, the SCD30 shows only very little drift over time, such that an FRC is only required once or twice per year (or after installing the SCD30 sensor into some device as it might have experienced some mechanical stress).

To perform the FRC, the CO_2 monitor simply needs to be placed outside somewhere where it is exposed to fresh air. Note that the sensor itself should not experience strong winds as this would deteriorate the measurements. The whole sensor should be in thermodynamic equilibrium before starting the FRC so it is best to operate it for a time of approximately 10 min before starting the FRC (for more details about the code, see Section 6).

5 Assembly

The CO_2 monitor can be assembled in various ways, first we will restrict ourselves to the case of a simple prototype design on a breadboard as shown in Fig. 5. The connection between the NodeMCU (with the ESP8266) and the SCD30 sensor is as follows:

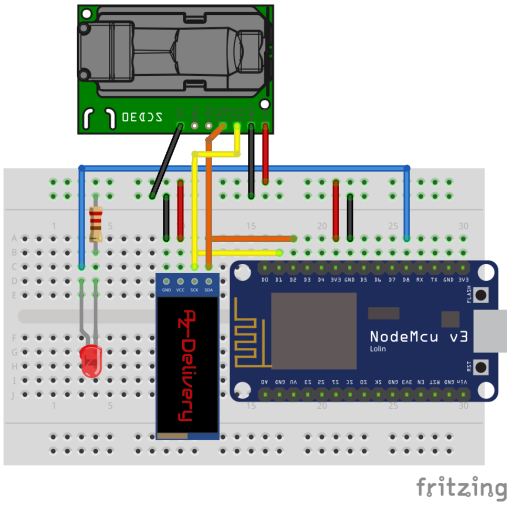

Figure 5: Schematic of a prototype of the CO_2 monitor (version 1).

NodeMCU		SCD30
`GND`	$\longrightarrow$	`GND`
3.3 V	$\longrightarrow$	`VIN`
`D2/GPIO4`	$\longrightarrow$	`RX/SDA`
`D1/GPIO5`	$\longrightarrow$	`TX/SCL`
`GND`	$\longrightarrow$	`SEL`

The NodeMCU ESP8266 board then needs to be connected to the OLED display as follows:

NodeMCU		OLED display
`GND`	$\longrightarrow$	`GND`
3.3 V	$\longrightarrow$	`VCC`
`D2/GPIO4`	$\longrightarrow$	`SDA`
`D1/GPIO5`	$\longrightarrow$	`SCL`

It is of course also possible to directly connect the respective `SDA` and `SCL` pins of the OLED and the SCD30, as shown in Fig. 5, instead of connecting those pins between the SCD30 and the NodeMCU. The red LED is connected with its anode, the longer leg, to pin `D8/GPIO15` of the NodeMCU and with its cathode, the shorter leg, via a 220 Ω resistor (to limit the current) to ground.

As discussed in Section 2, different assemblies of the CO_2 monitor are

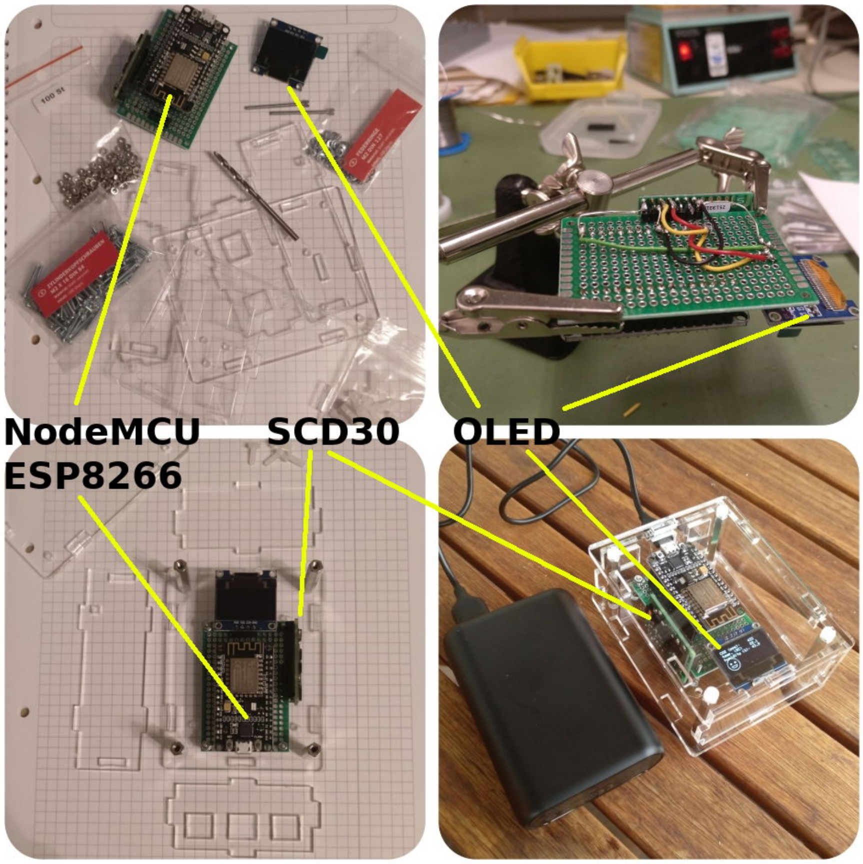

Figure 6: CO_2 monitor, version 3, in various stages of the assembly process. As compared to version 2, see Fig. 2, a larger OLED display is used, the SCD30 is mounted in a different way, and a slightly smaller stripboard is used.

possible. Figure 6 shows what we refer to as *version 3* in various stages of the assembly process. Compared to version 2, a larger OLED display is used, which requires to updates its size at the beginning of the program code, see Listing 3. As can be seen, the breadboard from version 1 is replaced by a stripboard onto which the necessary components need to be soldered. The casing is the same as in version 2 (which was originally intended to be used for an Arduino UNO). The side wall closest to the SCD30 has been omitted to ensure that the gas atmosphere inside the casing is the same as outside (a few holes drilled into that wall should have had the same effect). A portable battery charger, a power bank, is used as a power supply allowing for mobile usage of the CO_2 monitor across the day.

6 The program code

Arduino is used as programming language in this project due to its widespread usage and large numbers of libraries available for various hardware components. The Arduino IDE library manager allows to directly install a proper Arduino library for the SCD30. Alternatively, the library is available as a GitHub repository [42]. For a tutorial on how to install libraries within the Arduino IDE, see Ref. [56]. As for the NodeMCU and the OLED display, the Arduino IDE library manager is able to provide the required libraries.

The source code for the CO_2 monitor as described in this paper is available on GitHub [57], in order to be able to update and extend it. Nevertheless, we have also included the code in this paper, to provide a complete description of the project. At the very beginning of the code, some switches are set defining the general behavior of the program, as can be seen in Listing 1.

```
// -------------------------------------------------------------------
// Some switches defining general behaviour of the program
// -------------------------------------------------------------------
#define WIFI_ENABLED false       // set to true if WiFi is desired,
                                 // otherwise corresponding code is not compiled
#if WIFI_ENABLED
  #define WIFI_WEBSERVER true    // website in your WiFi for data and data logger
  #define WIFI_MQTT false        // activate MQTT integration in your WiFi
#endif
#define DEBUG true               // activate debugging
                                 // true:  print info + data to serial monitor
                                 // false: serial monitor is not used
#define DISPLAY_OLED             // OLED display
//#define DISPLAY_LCD            // LCD display
#define SEND_VCC false
// -------------------------------------------------------------------
```

Listing 1: General behavior of the program is set via some switches.

The `include` statements to import the required libraries are shown in Listing 2. The `Adafruit_GFX.h` and `Adafruit_SSD1306.h` libraries are used for an OLED display (if one is connected) and are required to be installed via the library manager of the Arduino IDE beforehand (alternatively, they are also available on GitHub [58] for manual installation). The `LiquidCrystal_I2C.h` library is used for the LCD display (if one is connected) and is also required to be installed via the library manager (or directly from GitHub [59]). Note that the OLED and/or LCD display size needs to be set correctly and can vary. The `SparkFun_SCD30_Arduino_Library.h` also needs to be installed via the library manager (or manually from the GitHub repository [42]).

```
// -------------------------------------------------------------------
// Import all required libraries
```

```
// -------------------------------------------------------------------
#include <Wire.h>                  // for I2C communication
#ifdef DISPLAY_OLED
  #include <Adafruit_GFX.h>        // for writing to display
  #include <Adafruit_SSD1306.h>    // for writing to display
#endif
#ifdef DISPLAY_LCD
  #include <LiquidCrystal_I2C.h>
#endif
#include "SparkFun_SCD30_Arduino_Library.h"

#include <ESP8266WiFi.h>           // also allows to explicitely turn WiFi off
#if WIFI_ENABLED
  #if WIFI_WEBSERVER
    #include <Hash.h>              // for SHA1 algorith (for Font Awesome)
    #include <ESPAsyncTCP.h>
    #include <ESPAsyncWebServer.h>

    #include "Webpageindex.h"      // webpage content, same folder as .ino file
  #endif

  #if WIFI_MQTT
    #include <PubSubClient.h>      // allows to send and receive MQTT messages

    //add local MQTT server IP here.
    IPAddress mqttserver(192, 168, 1, 100);
  #endif
  // Replace with your network credentials
  const char* ssid       = "ENTER_SSID";
  const char* password   = "ENTER_PASSWORD";
  const char* deviceName = "ENTER_ESP_DEVICE_NAME";
#endif
// -------------------------------------------------------------------
```

Listing 2: Load required libraries.

A switch is included in the header of the code allowing to enable or disable WiFi capabilities (by setting the variable `WIFI_ENABLED` respectively to `true` or `false`). The libraries required for using WiFi are only included if the corresponding switch is set to `true`. In this example, we decided to use the `ESPAsyncWebServer` [60], based on `ESPAsyncTCP` [61], for a webserver supposed to run on the ESP8266 because asynchronous networks, as provided by these two libraries, allow us to handle more than just one connection at a time (which is important if used in a classroom environment). During the time of writing this article, these libraries require manual installation, i.e. getting a `zip` file from the GitHub repositories and include those zip files manually as libraries in the Arduino IDE.

Hardware configurations, including the size of the display used, and some global constants are set after the include statements, as shown in Listing 3.

```
// -------------------------------------------------------------------
// Hardware configurations and some global constants
```

```
// -----------------------------------------------------------------
#define CO2_THRESHOLD1 600
#define CO2_THRESHOLD2 1000
#define CO2_THRESHOLD3 1500

#define WARNING_DIODE_PIN D8      // NodeMCU pin for red LED

#define MEASURE_INTERVAL 10       // seconds, minimum: 2

#define SCREEN_WIDTH 128          // OLED display width in pixels
#define SCREEN_HEIGHT 32          // OLED display height in pixels

const int lcdColumns  = 20;       // LCD: number of columns
const int lcdRows     = 4;        // LCD: number of rows

#define OLED_RESET LED_BUILTIN    // OLED reset pin, 4 is default
                                  // -1 if sharing Arduino reset pin
                                  // using NodeMCU, it is LED_BUILTIN

const float TempOffset = 5;       // temperature offset of the SCD30
                                  // 0 is default value
                                  // 5 is used in SCD30-library example
                                  // 5 also works for most of my devices

const int altitudeOffset = 300;   // altitude of place of operation in meters
                                  // Stuttgart: approx 300; Uni Stuttgart: approx 500; Lohne: 67

// update scd30 readings every MEASURE_INTERVAL seconds
const long interval = MEASURE_INTERVAL*1000;

// use "unsigned long" for variables that hold time
//  --> value will quickly become too large for an int
// store last time scd30 was updated
unsigned long previousMilliseconds = 0;

// switch to perform a forced recalibration
// should only be done once in a while and only when outside
bool DO_FORCED_RECALIBRATION = false;
// -----------------------------------------------------------------
```

Listing 3: Set some configurations.

Due to the complexity of the code, we decided to encapsulate certain parts in separate functions and use the technique of function prototyping and declaration. The function prototypes are shown in Listing 4.

```
// -----------------------------------------------------------------
// Function prototypes (not needed in Arduino, but good practice imho)
// -----------------------------------------------------------------
void airSensorSetup();
void forced_recalibration();
void printToSerial( float co2, float temperature, float humidity);
#ifdef DISPLAY_OLED
  void printToOLED( float co2, float temperature, float humidity);
  void printEmoji( float value );
#endif
#ifdef DISPLAY_LCD
  void printToLCD( float co2, float temperature, float humidity);
  void scrollLCDText( int row, String message, int delayTime );
```

```
#endif
// ----------------------------------------------------------------
```

Listing 4: Function prototypes.

Following the function prototypes, hardware declarations are executed, as shown in Listing 5.

```
// ----------------------------------------------------------------
// Hardware declarations
// ----------------------------------------------------------------
#ifdef DISPLAY_OLED
  // Declaration for an SSD1306 display connected to I2C (SDA, SCL pins)
  Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
#endif
#ifdef DISPLAY_LCD
  // run I2C scanner if LCD address is unknown
  LiquidCrystal_I2C lcd(0x27, lcdColumns, lcdRows);
#endif

SCD30 airSensor;
// ----------------------------------------------------------------
```

Listing 5: Hardware declarations.

To display the values measured by the SCD30 sensor on a website, we use global variables in the code, as shown in Listing 6. The complete html code for the website is loaded via including it as a library.

```
#if WIFI_ENABLED
  // temperature, humidity, CO2 for web-page, updated in loop()
  float temperature_web = 0.0;
  float humidity_web    = 0.0;
  float co2_web         = 0.0;

  #if WIFI_WEBSERVER
    // create AsyncWebServer object on port 80 (port 80 for http)
    AsyncWebServer server(80);
  #endif
  #if WIFI_MQTT
    // declare (initialize) object of class WiFiClient (to establish
    connection to IP/port)
    WiFiClient espClient;
    // declare (initialize) object of class PubSubClient
    // input: constructor of previously defined WiFiClient
    PubSubClient mqttClient(espClient);
    // message to be published to mqtt topic
    char mqttMessage[10];
  #endif
#endif
```

Listing 6: Prepare website.

The code for the webpage itself is shown in Listing 7.

```
//
// PROGMEM stores data in flash memory, default is storing in SRAM
//   --> usually, only worth to be used for large block of data
// R infront of string indicates a RAW string literal
```

```
//    --> no need to escape linebreaks, quotationmarks etc.
//    --> allows to put full html-website into variable
//    --> beginning and end of RAW string literal indicated by
//        =====( ... )=====
//
const char MAIN_page[] PROGMEM = R"=====(
<!doctype html>
<html>

<head>
  <title>CO2 Monitor</title>
  <!-- very helpful reference: https://www.w3schools.com -->
  <meta charset="utf-8">
  <!-- make webpage fit to your browser, not matter what OS or browser (also adjusts font sizes) -->
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <!-- prevent request on favicon, otherwise ESP receives favicon request every time web server is accessed -->
  <link rel="icon" href="data:,">
  <!-- load Font Awesome, get integrity and url here: https://fontawesome.com/account/cdn -->
  <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.14.0/css/all.css" integrity="sha384-HzLeBuhoNPvSl5KYnjx0BT+WB0QEEqLprO+NBkkk5gbc67FTaL7XIGa2w1L0Xbgc" crossorigin="anonymous">
  <!-- load chart.js library, this could also copied to esp8266 for usuage without internet connection -->
  <script src = "https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.7.3/Chart.min.js"></script>
  <style>
    canvas{
      -moz-user-select: none;
      -webkit-user-select: none;
      -ms-user-select: none;
    }
    html {
     font-family      : Arial;
     display          : inline-block;
     margin           : 0px auto;
     text-align       : center;
    }

    /* data table styling */
    #dataTable {
      font-family     : "Trebuchet MS", Arial, Helvetica, sans-serif;
      border-collapse : collapse;
      width           : 100%;
    }
    #dataTable td, #dataTable th {
      border          : 1px solid #ddd;
      padding         : 8px;
    }
    #dataTable tr:nth-child(even){
      background-color: #f2f2f2;
    }
    #dataTable tr:hover {
      background-color: #ddd;
    }
    #dataTable th {
      padding-top     : 12px;
      padding-bottom  : 12px;
      text-align      : left;
      background-color: #4CAF50;
```

```
      color            : white;
    }
  </style>
</head>

<body>
  <div style="text-align:center;">
    <p style="font-size:20px">
      <b>CO<sub>2</sub> Monitor: data logger (using Chart.js)</b>
      <button type="button" onclick="downloadData()">Download data</button>
    </p>
    <p>
      <i class="fab fa-github" style="font-size:1.0rem;color:black;"></i>
      <a href="https://github.com/alfkoehn/CO2_monitor" target="_blank" style="font-size:1.0rem;">Documentation &amp; code on GitHub</a>
    </p>
    <p>
      <i class="fab fa-twitter" style="font-size:1.0rem;color:#1DA1F2;"></i>
      <span style="font-size:1.0rem;">Contact via twitter: </span>
      <a href="https://twitter.com/formbar" target="_blank" style="font-size:1.0rem;">&#64;formbar</a>
    </p>
  </div>
  <br>
  <div class="chart-container" position: relative; height:350px; width:100%">
      <canvas id="Chart1" width="400" height="400"></canvas>
  </div>
  <br>
  <div class="chart-container" position: relative; height:350px; width:100%">
      <canvas id="Chart2" width="400" height="400"></canvas>
  </div>
  <br>
  <div>
    <table id="dataTable">
      <tr>
        <th><i class="far fa-clock"></i> Time</th>
        <th><i class="fas fa-head-side-cough" style="color:#ffffff;"></i> CO2 concentration in ppm</th>
        <th><i class="fas fa-thermometer-half" style="color:#ffffff;"></i> Temperaure in &deg;C</th>
        <th><i class="fas fa-tint" style="color:#ffffff;"></i> Humidity in %</th>
      </tr>
    </table>
  </div>
<br>
<br>

<script>

  // arrays for data values, will be dynamically filled
  // if length exceeds threshold, first (oldest) element is deleted
  var CO2values       = [];
  var Tvalues         = [];
  var Hvalues         = [];
  var timeStamp       = [];
  var maxArrayLength  = 1000;

  // update intervall for getting new data in milliseconds
  var updateIntervall = 10000;
```

```
// Graphs visit: https://www.chartjs.org
// graph for CO2 concentration
var ctx = document.getElementById("Chart1").getContext('2d');
var Chart1 = new Chart(ctx, {
  type: 'line',
  data: {
    labels: timeStamp,  //Bottom Labeling
    datasets: [{
      label             : "CO2 concentration",
      fill              : 'origin',                      // 'origin': fill area to x-axis
      backgroundColor : 'rgba( 243, 18, 156 , .5)', // point fill color
      borderColor      : 'rgba( 243, 18, 156 , 1)',  // point stroke color
      data              : CO2values,
    }],
  },
  options: {
    title: {
      display          : false,
      text             : "CO2 concentration"
    },
    maintainAspectRatio: false,
    elements: {
      line: {
        tension        : 0.5 //Smoothening (Curved) of data lines
      }
    },
    scales: {
      yAxes: [{
        display        : true,
        position       : 'left',
        ticks: {
          beginAtZero :false,
          precision    : 0,
          fontSize     :16
        },
        scaleLabel: {
          display      : true,
          // unicode for subscript:   u+208x,
          //          for superscript: u+207x
          labelString : 'CO\u2082 in ppm',
          fontSize     : 20
        },
      }]
    }
  }
});
// temperature and humidity graph
var ctx2 = document.getElementById("Chart2").getContext('2d');
var Chart2 = new Chart(ctx2, {
  type: 'line',
  data: {
    labels: timeStamp,  //Bottom Labeling
    datasets: [{
      label             : "Temperature",
      fill              : false,                         // fill area to xAxis
      backgroundColor : 'rgba( 243, 156, 18 , 1)',  // marker color
      borderColor      : 'rgba( 243, 156, 18 , 1)',  // line Color
      yAxisID          : 'left',
      data              : Tvalues,
    }, {
```

```
      label           : "Humidity",
      fill            : false,                    // fill area to xAxis
      backgroundColor : 'rgba(104, 145, 195, 1)',   // marker color
      borderColor     : 'rgba(104, 145, 195, 1)',   // line Color
      data            : Hvalues,
      yAxisID         : 'right',
    }],
  },
  options: {
    title: {
      display         : false,
      text            : "CO2 Monitor"
    },
    maintainAspectRatio: false,
    elements: {
      line: {
        tension       : 0.4                       // smoothening (bezier curve tension)
      }
    },
    scales: {
      yAxes: [{
        id            : 'left',
        position      : 'left',
        scaleLabel: {
          display     : true,
          labelString : 'Temperature in \u00B0C',
          fontSize    : 20
        },
        ticks: {
          suggestedMin: 18,
          suggestedMax: 30,
          fontSize    : 16
        }
      }, {
        id            : 'right',
        position      : 'right',
        scaleLabel: {
          display     : true,
          labelString : 'Humidity in %',
          fontSize    : 20
        },
        ticks: {
          suggestedMin: 40,
          suggestedMax: 70,
          fontSize    : 16
        }
      }]
    }
  }
});

// function to dynamically updating graphs
// much more efficient than replotting every time
function updateCharts() {
  // update datasets to be plotted
  Chart1.data.datasets[0].data = CO2values;
  Chart2.data.datasets[0].data = Tvalues;
  Chart2.data.datasets[1].data = Hvalues;
  // update the charts
  Chart1.update();
  Chart2.update();
```

```
};

// function to download data arrays into csv-file
function downloadData() {
  // build array of strings with data to be saved
  var data = [];
  for ( var ii=0 ; ii < CO2values.length ; ii++ ){
    data.push( [ timeStamp[ii],
                 Math.round(CO2values[ii]).toString(),
                 Tvalues[ii].toString(),
                 Hvalues[ii].toString()
               ]);
  }

  // build String containing all data to be saved (csv-formatted)
  var csv = 'Time,CO2 in ppm,Temperature in Celsius,Humidity in percent\n';
  data.forEach(function(row) {
    csv += row.join(',');
    csv += "\n";
  });

  // save csv-string into file
  // create a hyperlink element (defined by <a> tag)
  var hiddenElement     = document.createElement('a');
  // similar functions: encodeURI(), encodeURIComponent() (escape() not recommended)
  hiddenElement.href    = 'data:text/csv;charset=utf-8,'+encodeURI(csv);
  hiddenElement.target  = '_blank';
  hiddenElement.download= 'CO2monitor.csv';
  hiddenElement.click();
};

// ajax script to get data repetivitely

setInterval(function() {
  // call a function repetitively, intervall set by variable <updateIntervall>
  getData();
}, updateIntervall);

function getData() {
  var xhttp           = new XMLHttpRequest();

  // onreadystatechange property defines a function to be executed when the readyState changes
  xhttp.onreadystatechange = function() {

    // "readyState" property holds the status of the "XMLHttpRequest"
    //          values: 0   --> request not initialized
    //                  1   --> server connection established
    //                  2   --> request received
    //                  3   --> processing request
    //                  4   --> request finished and response is ready
    // "status" values: 200 --> "OK"
    //                  403 --> "Forbidden"
    //                  404 --> "Page not found"
    // "this" keyword always refers to objects it belongs to
    if (this.readyState == 4 && this.status == 200) {
      //Push the data in array

      // Date() creates a Date object
```

```
        // toLocaleTimeString() returns time portion of Date object as
    string
        var time = new Date().toLocaleTimeString();

        // read-only XMLHttpRequest property responseText returns
        // text received from a server following a request being sent
        var txt = this.responseText;

        // data from webserver is always a string, parsing with JSON.parse()
        // to let data beome a JavaScrip object
        var obj = JSON.parse(txt);

        // add elements to arrays
        // push() methods adds new items to end of array, returns new length
        CO2values.push(obj.COO);
        Tvalues.push(obj.Temperature);
        Hvalues.push(obj.Humidity);
        timeStamp.push(time);

        // if array becomes too long, delete oldest element to not overload
    graph
        // also delete first row of data table
        if (CO2values.length > maxArrayLength) {
          // shift() method to delete first element
          CO2values.shift();
          Tvalues.shift();
          Hvalues.shift();
          timeStamp.shift();

          // HTMLTableElement.deleteRow(index), index=-1 for last element
          document.getElementById("dataTable").deleteRow(-1);
        }

        // update graphs
        updateCharts();

        // update data table
        var table       = document.getElementById("dataTable");
        var row         = table.insertRow(1); //Add after headings
        var cell1       = row.insertCell(0);
        var cell2       = row.insertCell(1);
        var cell3       = row.insertCell(2);
        var cell4       = row.insertCell(3);
        cell1.innerHTML = time;
        cell2.innerHTML = Math.round(obj.COO);
        cell3.innerHTML = obj.Temperature;
        cell4.innerHTML = obj.Humidity;
      }
    };
    xhttp.open("GET", "readData", true); //Handle readData server on ESP8266
    xhttp.send();
  }

</script>
</body>
</html>
)=====";
```

Listing 7: Code for the webpage.

As usual, the function declarations are all located at the end of the code but are briefly discussed first before coming to the main `setup()` and `loop()` functions. The function to print the data obtained from the SCD30 to the serial console is shown in Listing 8. Since it is possible to use an OLED and/or LCD display to show the measured data, a separate function for each case is included in the code, see Listing 9 and Listing 10

```
void printToSerial( float co2, float temperature, float humidity) {
  Serial.print("co2(ppm):");
  Serial.print(co2, 1);
  Serial.print(" temp(C):");
  Serial.print(temperature, 1);
  Serial.print(" humidity(%):");
  Serial.print(humidity, 1);
  Serial.println();
}
```

Listing 8: Function which prints data to the serial console.

```
#ifdef DISPLAY_OLED
void printToOLED( float co2, float temperature, float humidity) {
  int
    x0, x1;           // to align output on OLED display vertically

  x0  = 0;
  x1  = 84;

  display.clearDisplay();
  display.setCursor(x0,5);
  display.print("CO2 (ppm):");
  display.setCursor(x1,5);
  // for floats, 2nd parameter in display.print sets number of decimals
  display.print(co2, 0);

  display.setCursor(x0,15);
  display.print("temp. ( C)");
  display.setCursor(x0+7*6,15);
  display.cp437(true);  // enable full 256 char 'Code Page 437' font
  display.write(248);   // degree symbol
  display.setCursor(x1,15);
  display.print(temperature, 1);

  display.setCursor(x0,25);
  display.print("humidity (%):");
  display.setCursor(x1,25);
  display.print(humidity, 1);

  display.display();
}
#endif
```

Listing 9: Function which prints data to an OLED display.

```
#ifdef DISPLAY_LCD
void printToLCD( float co2, float temperature, float humidity) {

  byte degreeSymbol[8] = {
    0b01100, 0b10010, 0b10010, 0b01100,
    0b00000, 0b00000, 0b00000, 0b00000
```

```
  };
  // allocate custom char to a location
  lcd.createChar(0, degreeSymbol);

  //int waitTime  = 2000;
  lcd.clear();
  //DrawYoutube();
  //delay(waitTime);

  // print co2 concentration (1st line, i.e. row 0)
  lcd.setCursor(0,0);
  lcd.print("CO2 in ppm");
  // make output right-aligned
  lcd.setCursor( (lcdColumns - (int(log10(co2))+1)), 0);
  lcd.print(int(round(co2)));

  // print temperature (2nd line, i.e. row 1)
  lcd.setCursor(0,1);
  lcd.print("Temp. in ");
  lcd.write(0);
  lcd.print("C");
  // make output right-aligned
  lcd.setCursor( (lcdColumns - (int(log10(temperature))+1)), 1);
  lcd.print(int(round(temperature)));

  // print humidity (3nd line, i.e. row 2)
  lcd.setCursor(0,2);
  lcd.print("Humidity in %");
  // make output right-aligned
  lcd.setCursor( (lcdColumns - (int(log10(humidity))+1)), 2);
  lcd.print(int(round(humidity)));

  //delay(waitTime);
}
#endif
```

Listing 10: Function which prints data to an LCD display.

Depending on the the CO_2 concentration, the OLED display shows an emoji with the level of happiness being correlated to the value of the CO_2 concentration. The corresponding function to draw that face is given in Listing 11. For the LCD display, a warning is shown if the CO_2 concentration is too high, the corresponding code is shown in Listing 12.

```
#ifdef DISPLAY_OLED
void printEmoji( float value ) {
  // syntax for functions used to draw to OLED:
  // display.drawBitmap(x, y, bitmap data, bitmap width, bitmap height,
    color)
  // display.drawCircle(x, y, radius, color)

  float start_angle,    // used for smiley mouth
        end_angle,      // used for smiley mouth
        i;              // used for smiley mouth
  int   smile_x0,
        smile_y0,
        smile_r,
        emoji_r,
        emoji_x0,
```

```
        emoji_y0,
        eye_size;

  emoji_r   = SCREEN_HEIGHT/4;
  if (SCREEN_HEIGHT == 32) {
    emoji_x0  = SCREEN_WIDTH - (1*emoji_r+1);
    emoji_y0  = emoji_r*3-1;
    eye_size  = 1;
  } else if (SCREEN_HEIGHT == 64) {
    emoji_x0  = emoji_r;
    emoji_y0  = emoji_r*3-1;
    eye_size  = 2;
  }

  bool  plot_all;       // if set, plots all emojis (makes only sense for larger oled)

  plot_all  = false;
  if (int(value) == 0) {
    plot_all  = true;
  }

  if (value < CO2_THRESHOLD1){
    // very happy smiley face

    display.drawCircle(emoji_x0*1, emoji_y0, emoji_r, WHITE);

    start_angle = 20./180*PI;
    end_angle   = 160./180*PI;
    smile_r   = emoji_r/2;
    smile_x0  = emoji_x0*1;
    smile_y0  = emoji_y0+emoji_r/6;
    for (i = start_angle; i < end_angle; i = i + 0.05) {
      display.drawPixel(smile_x0 + cos(i) * smile_r, smile_y0 + sin(i) * smile_r, WHITE);
    }

    display.drawLine(smile_x0+cos(start_angle)*smile_r, smile_y0+sin(start_angle)*smile_r,
                     smile_x0+cos(end_angle)*smile_r, smile_y0+sin(end_angle)*smile_r,
                     WHITE);

    // draw eyes
    display.fillCircle(emoji_x0*1-emoji_r/2/4*3, smile_y0-emoji_r/3, eye_size, WHITE);
    display.fillCircle(emoji_x0*1+emoji_r/2/4*3, smile_y0-emoji_r/3, eye_size, WHITE);
  }
  if ((value >= CO2_THRESHOLD1 && value < CO2_THRESHOLD2) || (plot_all == true)) {
    // happy smiley face

    if (SCREEN_HEIGHT == 32) {
      display.drawCircle(emoji_x0, emoji_y0, emoji_r, WHITE);
    } else if (SCREEN_HEIGHT == 64) {
      display.drawCircle(emoji_x0 + 2*emoji_r, emoji_y0, emoji_r, WHITE);
    }

    // draw mouth
    if (SCREEN_HEIGHT == 32) {
      smile_x0  = emoji_x0;
```

```
    } else if (SCREEN_HEIGHT == 64) {
      smile_x0  = emoji_x0 + 2*emoji_r;
    }
    start_angle = 20./180*PI;
    end_angle   = 160./180*PI;
    smile_r   = emoji_r/2;
    smile_y0  = emoji_y0+emoji_r/6;
    for (i = start_angle; i < end_angle; i = i + 0.05) {
      display.drawPixel(smile_x0 + cos(i) * smile_r, smile_y0 + sin(i) * smile_r, WHITE);
    }

    // draw eyes
    display.fillCircle(smile_x0-emoji_r/2/4*3, smile_y0-emoji_r/3, eye_size, WHITE);
    display.fillCircle(smile_x0+emoji_r/2/4*3, smile_y0-emoji_r/3, eye_size, WHITE);
  }
  if ((value >= CO2_THRESHOLD2 && value < CO2_THRESHOLD3) || (plot_all == true)) {
    // not so happy smiley face

    if (SCREEN_HEIGHT == 32) {
      display.drawCircle(emoji_x0, emoji_y0, emoji_r, WHITE);
    } else if (SCREEN_HEIGHT == 64) {
      display.drawCircle(emoji_x0 + 4*emoji_r, emoji_y0, emoji_r, WHITE);
    }

    // draw mouth
    if (SCREEN_HEIGHT == 32) {
      smile_x0  = emoji_x0;
    } else if (SCREEN_HEIGHT == 64) {
      smile_x0  = emoji_x0 + 4*emoji_r;
    }
    display.drawLine(smile_x0-emoji_r/2/4*3, emoji_y0+emoji_r/2,
                     smile_x0+emoji_r/2/4*3, emoji_y0+emoji_r/2,
                     WHITE);

    // draw eyes
    display.fillCircle(smile_x0-emoji_r/2/4*3, smile_y0-emoji_r/3, eye_size, WHITE);
    display.fillCircle(smile_x0+emoji_r/2/4*3, smile_y0-emoji_r/3, eye_size, WHITE);
  }
  if ((value >= CO2_THRESHOLD3) || (plot_all == true)) {
    // sad smiley face

    if (SCREEN_HEIGHT == 32) {
      display.drawCircle(emoji_x0, emoji_y0, emoji_r, WHITE);
    } else if (SCREEN_HEIGHT == 64) {
      display.drawCircle(emoji_x0 + 6*emoji_r-1, emoji_y0, emoji_r, WHITE);
    }

    // draw mouth
    if (SCREEN_HEIGHT == 32) {
      smile_x0  = emoji_x0;
    } else if (SCREEN_HEIGHT == 64) {
      smile_x0  = emoji_x0 + 6*emoji_r;
    }
    start_angle = 200./180*PI;
    end_angle   = 340./180*PI;
    smile_r   = emoji_r/2;
```

```
    smile_y0  = emoji_y0+emoji_r/6;
    for (i = start_angle; i < end_angle; i = i + 0.05) {
      display.drawPixel(smile_x0 + cos(i) * smile_r, smile_y0+emoji_r/2 +
    sin(i) * smile_r, WHITE);
    }

    // draw eyes
    display.fillCircle(smile_x0-emoji_r/2/4*3, smile_y0-emoji_r/3, eye_size,
     WHITE);
    display.fillCircle(smile_x0+emoji_r/2/4*3, smile_y0-emoji_r/3, eye_size,
     WHITE);
  }
  display.display();
}
#endif
```

Listing 11: Function which prints smileys to the OLED display depending on the value of the CO_2 concentration.

```
#ifdef DISPLAY_LCD
// Parameters:
//   row:        where text will be printed
//   message:    text to scroll
//   delayTime:  time between character shifting
// inspired by https://randomnerdtutorials.com/esp32-esp8266-i2c-lcd-arduino
    -ide/
void scrollLCDText( int row, String message, int delayTime ){
  // add whitespaces equal to no. LCD-columns at beginning of string
  for (int i=0; i<lcdColumns ; ++i) {
    message = " "+message;
  }
  message = message+" ";
  // emulate the scrolling by printing substrings sequentially
  for (int pos=0 ; pos<message.length(); ++pos){
    lcd.setCursor(0,row);
    lcd.print(message.substring(pos, pos+lcdColumns));
    delay(delayTime);
  }
}
#endif
```

Listing 12: Function which prints a warning to the LCD display depending on the value of the CO_2 concentration.

Listing 13 shows the `setup` function of the code, where the serial monitor is initialized, followed by the diode, optionally the WiFi, the OLED display, and then the SCD30. Finally, the webserver and the functions required to update the data on the webpage are prepared.

```
void setup(){
  if (DEBUG == true) {
    // initialize serial monitor at baud rate of 115200
    Serial.begin(115200);
    delay(1000);
    Serial.println("Using SCD30 to get: CO2 concentration, temperature,
    humidity");
  }

```

```
  // initialize I2C
  Wire.begin();

  // initialize LED pin as an output
  pinMode(WARNING_DIODE_PIN, OUTPUT);

#if WIFI_ENABLED
  #if WIFI_MQTT
  // configure mqtt server details, after that client is ready
  // create connection to mqtt broker
  mqttClient.setServer(mqttserver, 1883);
  #endif
  /* Explicitly set ESP8266 to be a WiFi-client, otherwise, it would, by
     default, try to act as both, client and access-point, and could cause
     network-issues with other WiFi-devices on your WiFi-network. */
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);     // connect to Wi-Fi
  if (DEBUG == true)
    Serial.println("Connecting to WiFi");
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    if (DEBUG == true)
      Serial.print(".");
  }
  IPAddress ip = WiFi.localIP();
  if (DEBUG == true)
    Serial.println(ip);

  #if WIFI_WEBSERVER
    // ------------------------------------------------------------------
    // This is executed when you open the IP in browser
    // ------------------------------------------------------------------
    server.on("/", HTTP_GET, [](AsyncWebServerRequest *request) {
      // note: do NOT load MAIN_page into a String variable
      //       this might not work (probably too large)
      request->send_P(200, "text/html", MAIN_page );
    });

    // this page is called by java Script AJAX
    server.on("/readData", HTTP_GET, [](AsyncWebServerRequest *request) {
      // putting all values into one big string
      // inspiration: https://circuits4you.com/2019/01/11/nodemcu-esp8266-arduino-json-parsing-example/
      String data2send = "{\"CO2\":\""+String(co2_web)
                        +"\", \"Temperature\":\""+String(temperature_web)
                        +"\", \"Humidity\":\""+ String(humidity_web) +"\"}";
      request->send_P(200, "text/plain", data2send.c_str());
    });
    // ------------------------------------------------------------------

    server.begin();
  #endif
#else
  WiFi.mode( WIFI_OFF );          // explicitely turn WiFi off
  WiFi.forceSleepBegin();         // explicitely turn WiFi off
  delay( 1 );                     // required to apply WiFi changes
  if (DEBUG == true)
    Serial.println("WiFi is turned off.");
#endif

#ifdef DISPLAY_OLED
```

```
  // SSD1306_SWITCHCAPVCC: generate display voltage from 3.3V internally
  if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { // Address 0x3C for 128x32
    if (DEBUG == true)
      Serial.println(F("SSD1306 allocation failed"));
    for(;;);                        // don't proceed, loop forever
  }
  display.display();                // initialize display
                                    // library will show Adafruit logo
  delay(2000);                      // pause for 2 seconds
  display.clearDisplay();           // clear the buffer
  display.setTextSize(1);           // has to be set initially
  display.setTextColor(WHITE);      // has to be set initially

  // move cursor to position and print text there
  display.setCursor(2,5);
  display.println("CO2 monitor");
  display.println("twitter.com/formbar");
  #if WIFI_ENABLED
    display.println(ip);
  #else
    display.println("WiFi disabled");
  #endif

  // write previously defined emojis to display
  if (SCREEN_HEIGHT == 32) {
    printEmoji(400);
    delay(2000);
    printEmoji(600);
    delay(2000);
    printEmoji(1200);
    delay(2000);
    printEmoji(1800);
    delay(2000);
  } else {
    printEmoji(0);
  }

  display.display();                // write display buffer to display
#endif

#ifdef DISPLAY_LCD
  lcd.init();                       // initialize LCD
  lcd.backlight();                  // turn on LCD backlight
  lcd.setCursor(0,0);               // set cursor to (column,row)
  lcd.print("WiFi connected");
  lcd.setCursor(0,1);
  lcd.print(ip);
  delay(2000);
#endif

  // turn warning LED on and off to test it
  digitalWrite(WARNING_DIODE_PIN, HIGH);
  delay(2000*2);
  digitalWrite(WARNING_DIODE_PIN, LOW);

  // initialize SCD30
  airSensorSetup();
}
```

Listing 13: General setup function.

The calibration and setup of the SCD30 is put into a separate function, shown in Listing 14. An additional function, given in Listing 15, performs the forced recalibration of the SCD30 discussed in Section 4.3.

```
void airSensorSetup(){

  bool autoSelfCalibration = false;

  // start sensor using the Wire port
  if (airSensor.begin(Wire) == false) {
    if (DEBUG == true)
      Serial.println("Air sensor not detected. Please check wiring. Freezing...");
    while (1)
      ;
  }

  // disable auto-calibration (import, see full documentation for details)
  airSensor.setAutoSelfCalibration(autoSelfCalibration);

  // SCD30 has data ready at maximum every two seconds
  // can be set to 1800 at maximum (30 minutes)
  //airSensor.setMeasurementInterval(MEASURE_INTERVAL);

  // altitude compensation in meters
  // alternatively, one could also use:
  //   airSensor.setAmbientPressure(pressure_in_mBar)
  delay(1000);
  airSensor.setAltitudeCompensation(altitudeOffset);

  float T_offset = airSensor.getTemperatureOffset();
  Serial.print("Current temp offset: ");
  Serial.print(T_offset, 2);
  Serial.println("C");

  // note: this value also depends on how you installed
  //       the SCD30 in your device
  airSensor.setTemperatureOffset(TempOffset);
}
```

Listing 14: Setup code for the SCD30.

```
void forced_recalibration(){
  // note: for best results, the sensor has to be run in a stable environment
  //       in continuous mode at a measurement rate of 2s for at least two
  //       minutes before applying the FRC command and sending the reference value
  // quoted from "Interface Description Sensirion SCD30 Sensor Module"

  String counter;

  int CO2_offset_calibration = 410;

  if (DEBUG == true){
    Serial.println("Starting to do a forced recalibration in 10 seconds");
  }

#ifdef DISPLAY_OLED
  display.clearDisplay();
```

```
  display.setCursor(0,0);
  display.println("Warning:");
  display.println("forced recalibration");
  display.display();

  for (int ii=0; ii<10; ++ii){
    counter = String(10-ii);
    display.setCursor(ii*9,20);
    display.print(counter);
    display.display();
    delay(1000);
  }
#endif

  airSensor.setForcedRecalibrationFactor(CO2_offset_calibration);

#ifdef DISPLAY_OLED
  delay(1000);
  display.clearDisplay();
  display.setCursor(0,0);
  display.println("Successfully recalibrated!");
  display.println("Only required ~once per year");
  display.display();
#endif

  delay(5000);
}
```

Listing 15: Function to perform a forced recalibration of the SCD30.

The main code, the `loop` function, is given in Listing 16. First, the data is obtained from the SCD30 sensor and then passed to a function outputting it to the serial monitor and then to another function, printing it on an OLED and/or LCD display. The data is then copied into the corresponding global variables to prepare the next update for the webpage. Finally, it is checked if the CO_2 concentration is above a critical threshold: a red LED indicates too high a value in our example, in addition some reaction on the OLED/LCD display is shown (one could also think of an acoustic signal and some visual change on the webpage).

```
void loop(){

  float
    co2_new,
    temperature_new,
    humidity_new;

  unsigned long currentMilliseconds;

  // get milliseconds passed since program started to run
  currentMilliseconds = millis();

  // forced recalibration requires 2 minutes of stable environment in
    advance
  if ((DO_FORCED_RECALIBRATION == true) && (currentMilliseconds > 120000)) {
    forced_recalibration();
    DO_FORCED_RECALIBRATION = false;
```

```
  }

  if (currentMilliseconds - previousMilliseconds >= interval) {
    // save the last time you updated the DHT values
    previousMilliseconds = currentMilliseconds;
#if SEND_VCC
    vdd = ESP.getVcc();
#endif

    if (airSensor.dataAvailable()) {
      // get updated data from SCD30 sensor
      co2_new         = airSensor.getCO2();
      temperature_new = airSensor.getTemperature();
      humidity_new    = airSensor.getHumidity();

      // print data to serial console
      if (DEBUG == true)
        printToSerial(co2_new, temperature_new, humidity_new);

      // print data to display
#ifdef DISPLAY_OLED
      printToOLED(co2_new, temperature_new, humidity_new);
      // print smiley with happiness according to CO2 concentration
      printEmoji( co2_new);
#endif
#ifdef DISPLAY_LCD
      printToLCD(co2_new, temperature_new, humidity_new);
      if (co2_new > CO2_THRESHOLD3)
        scrollLCDText( 3, "LUEFTEN", 250 );
#endif
      // if CO2-value is too high, issue a warning
      if (co2_new >= CO2_THRESHOLD3) {
        digitalWrite(WARNING_DIODE_PIN, HIGH);
      } else {
        digitalWrite(WARNING_DIODE_PIN, LOW);
      }
#if WIFI_ENABLED
      // updated values for webpage
      co2_web         = co2_new;
      temperature_web = temperature_new;
      humidity_web    = humidity_new;
#if WIFI_MQTT
      // boolean connect (clientID, [username, password])
      // see https://pubsubclient.knolleary.net/api
      mqttClient.connect(deviceName);
#if SEND_VCC
      sprintf(mqttMessage, "%d", vdd);
      // boolean publish (topic, payload)
      // publish message to the specified topic
      mqttClient.publish("esp-co2/co2/vcc", mqttMessage );
#endif
      mqttClient.publish("esp-co2/co2/hostname", deviceName );
      sprintf(mqttMessage, "%6.2f", co2_web);
      mqttClient.publish("esp-co2/co2/co2", mqttMessage );
      sprintf(mqttMessage, "%6.2f", temperature_web);
      mqttClient.publish("esp-co2/co2/temp", mqttMessage );
      sprintf(mqttMessage, "%6.2f", humidity_web);
      mqttClient.publish("esp-co2/co2/hum", mqttMessage );
#endif
#endif
    }
  }
```

```
  delay(100);
}
```

Listing 16: Main loop which is executed repeatedly.

Acknowledgments

The author is indebted to the efforts of the open-source software community.

References

[1] D. Twardella, W. Matzen, T. Lahrz, R. Burghardt, H. Spegel, L. Hendrowarsito, A. C. Frenzel, and H. Fromme. Effect of classroom air quality on students' concentration: results of a cluster-randomized cross-over experimental study. *Indoor Air*, 22:378, 2012, doi:10.1111/j.1600-0668.2012.00774.x.

[2] S. Gaihre, S. Semple, J. Miller, S. Fielding, and S. Turner. Classroom Carbon Dioxide Concentration, School Attendance, and Educational Attainment. *J. Sch. Health*, 85:569–574, 2014, doi:10.1111/josh.12183.

[3] D. G. Shendell, R. Prill, W. J. Fisk, M. G. Apte, D. Blake, and D. Faulkner. Associations between classroom co_2 concentrations and student attendance in washington and idaho. *Indoor Air*, 14:333, 2004, doi:10.1111/j.1600-0668.2004.00251.x.

[4] J. G. Allen, P. MacNaughton, U. Satish, S. Santanam, J. Vallarino, and J. D. Spengler. Associations of Cognitive Function Scores with Carbon Dioxide, Ventilation, and Volatile Organic Compound Exposures in Office Workers: A Controlled Exposure Study of Green and Conventional Office Environments. *Environmental Health Perspectives*, 6:805, 2016, doi:10.1289/ehp.1510037.

[5] A. Persily and L. de Jonge. Carbon dioxide generation rates for building occupants. *Indoor Air*, 2017, doi:10.1111/ina.12383.

[6] J. A. Lednicky, M. Lauzardo, Z. Hugh Fan, A. S. Jutla, T. B. Tilly, M. Gangwar, M. Usmani, S. N. Shankar, K. Mohamed, A. Eiguren-Fernandez, C. J. Stephenson, Alam. Md. M., M. A. Elbadry, J. C. Loeb, K. Subramaniam, T. B. Waltzek, K. Cherabuddi, J. G. Morris Jr., and C.-Y. Wu. Viable sars-cov-2 in the air of a hospital room with covid-19 patients. *medRxiv*, 2020, doi:10.1101/2020.08.03.20167395.

[7] L. Morawska and D. K. Milton. It is time to address airborne transmission of covid-19. *Clinical Infectious Diseases*, 2020, doi:10.1093/cid/ciaa939.

[8] M. A. Kohanski, L. J. Lo, and M. S. Waring. Review of indoor aerosol generation, transport, and control in the context of covid-19. *International Forum of Allergy & Rhinology*, 2020, doi:10.1002/alr.22661.

[9] Leonardo Setti, Fabrizio Passarini, Gianluigi De Gennaro, Pierluigi Barbieri, Maria Grazia Perrone, Massimo Borelli, Jolanda Palmisani, Alessia Di Gilio, Prisco Piscitelli, and Alessandro Miani. Airborne Transmission Route of COVID-19: Why 2 Meters/6 Feet of Inter-Personal Distance Could Not Be Enough. *International Journal of Environmental Research and Public Health*, 17(8), 2020, doi:10.3390/ijerph17082932.

[10] N. W. Furukawa, J. T. Brooks, and J. Sobel. Evidence supporting transmission of severe acute respiratory syndrome coronavirus 2 while presymptomatic or asymptomatic. *Emerg Infect Dis.*, 2020, doi:10.3201/eid2607.201595.

[11] Kaiyuan Sun, Wei Wang, Lidong Gao, Yan Wang, Kaiwei Luo, Lingshuang Ren, Zhifei Zhan, Xinghui Chen, Shanlu Zhao, Yiwei Huang, Qianlai Sun, Ziyan Liu, Maria Litvinova, Alessandro Vespignani, Marco Ajelli, Cécile Viboud, and Hongjie Yu. Transmission heterogeneities, kinetics, and controllability of sars-cov-2. *Science*, 2020, doi:10.1126/science.abe2424.

[12] Hua Qian, Te Miao, Li LIU, Xiaohong Zheng, Danting Luo, and Yuguo Li. Indoor transmission of sars-cov-2. *medRxiv*, 2020, doi:10.1101/2020.04.04.20053058.

[13] Jan M. Brauner, Sören Mindermann, Mrinank Sharma, David Johnston, John Salvatier, Tomáš Gavenčiak, Anna B. Stephenson, Gavin Leech, George Altman, Vladimir Mikulik, Alexander John Norman, Joshua Teperowski Monrad, Tamay Besiroglu, Hong Ge, Meghan A. Hartwick, Yee Whye Teh, Leonid Chindelevitch, Yarin Gal, and Jan Kulveit. Inferring the effectiveness of government interventions against COVID-19. *Science*, 2020, doi:10.1126/science.abd9338.

[14] Jeremy Howard, Austin Huang, Zhiyuan Li, Zeynep Tufekci, Vladimir Zdimal, Helene-Mari van der Westhuizen, Arne von Delft, Amy Price, Lex Fridman, Lei-Han Tang, Viola Tang, Gregory L. Watson,

Christina E. Bax, Reshama Shaikh, Frederik Questier, Danny Hernandez, Larry F. Chu, Christina M. Ramirez, and Anne W. Rimoin. An evidence review of face masks against COVID-19. *Proceedings of the National Academy of Sciences*, 118(4), 2021, doi:10.1073/pnas.2014564118.

[15] Rajat Mittal, Rui Ni, and Jung-Hee Seo. The flow physics of covid-19. *Journal of Fluid Mechanics*, 894:F2, 2020, doi:10.1017/jfm.2020.330.

[16] N. v. Doremalen, T. Bushmaker, D. H. Morris, M. G. Holbrook, A. Gamble, B. N. Williamson, A. Tamin, J. L. Harcourt, N. J. Thornburg, S. I. Gerber, J. O. Lloyd-Smith, E. de Witt, and V. J. Munster. Aerosol and Surface Stability of SARS-CoV-2 as Compared with SARS-CoV-1. *N. Engl. J. Med.*, 382:1564–1567, 2020, doi:10.1056/NEJMc2004973.

[17] X. Zhang, J. Zheng, Y. Yue, H. Lui, and J. Wang. Infection Risk Assessment of COVID-19 through Aerosol Transmission: a Case Study of South China Seafood Market. *Environmental science & technology*, 2020, doi:10.1021/acs.est.0c02895.

[18] Yuguo Li, Hua Qian, Jian Hang, Xuguang Chen, Ling Hong, Peng Liang, Jiansen Li, Shenglan Xiao, Jianjian Wei, Li Liu, and Min Kang. Evidence for probable aerosol transmission of sars-cov-2 in a poorly ventilated restaurant. *medRxiv*, 2020, doi:10.1101/2020.04.16.20067728.

[19] Shelly L Miller, William W Nazaroff, Jose L Jimenez, Atze Boerstra, Giorgio Buonanno, Stephanie J Dancer, Jarek Kurnitski, Linsey C Marr, Lidia Morawska, and Catherine Noakes. Transmission of sars-cov-2 by inhalation of respiratory aerosol in the skagit valley chorale superspreading event. *medRxiv*, 2020, doi:10.1101/2020.06.15.20132027.

[20] George A. Soper. The lessons of the pandemic. *Science*, 49:501, 1919, doi:10.1126/science.49.1274.501.

[21] A. Hartmann and M. Kriegel. Risk assessment of aerosols loaded with virus based on co_2-concentration. *Preprint server of the Technische Universität Berlin*, 2020, doi:10.14279/depositonce-10362.

[22] Rajesh K. Bhagat, M. S. Davies Wykes, Stuart B. Dalziel, and P. F. Linden. Effects of ventilation on the indoor spread of COVID-19. *Journal of Fluid Mechanics*, 903:F1, 2020, doi:10.1017/jfm.2020.720.

[23] A. Voß, A. Gritzki, and K. Bux. Infektionsschutzgerechtes Lüften – Hinweise und Maßnahmen in Zeiten der SARS-CoV-2-Epidemie.

Bundesanstalt für Arbeitsschutz und Arbeitsmedizin (BAuA), 2020, doi:10.21934/baua:fokus20200918.

[24] Federal Ministry of Labour and Social Affairs Germany. Website: Empfehlung der Bundesregierung "Infektionsschutzgerechtes Lüften" (version 2020-09-16). https://www.bmas.de/SharedDocs/Downloads/DE/Thema-Arbeitsschutz/infektionsschutzgerechtes-lueften.html.

[25] Robert Koch Institut. Präventionsmaßnahmen in Schulen während der COVID-19-Pandemie. Empfehlungen des Robert Koch-Instituts für Schulen. (version 2020-10-12). https://www.rki.de/DE/Content/InfAZ/N/Neuartiges_Coronavirus/Praevention-Schulen.pdf?__blob=publicationFile.

[26] Chun-Ru Du, Shun-Chih Wang, Ming-Chih Yu, Ting-Fang Chiu, Jann-Yuan Wang, Pei-Chun Chuang, Ruwen Jou, Pei-Chun Chan, and Chi-Tai Fang. Effect of ventilation improvement during a tuberculosis outbreak in underventilated university buildings. *Indoor Air*, 30(3):422–432, 2020, doi:10.1111/ina.12639.

[27] NALTIC Industrials, LLC. Aranet4 co_2 meter. https://naltic.com/aranet4-co2.html#.

[28] TFA Dostmann. AIRCO_2NTROL. https://www.tfa-dostmann.de/en/produkte/professional-usage-16/measuring-instruments-for-professional-usage/co2-measuring-instruments/.

[29] Wöhler Technik GmbH. CO2 Messgerät. https://www.woehler.de/shop/co2-messgeraet.html.

[30] B. Watzka and R. Girwidz. Ist die Luft zu schlecht zum Lernen? Nichtdispersive IR-CO2-Gassensoren im Physikunterricht. *Physik und Didaktik in Schule und Hochschule*, 10:22–33, 2011.

[31] Hochschule Trier Umwelt-Campus Birkenfeld. Website: Covid-19 Prävention: CO2-Messung und bedarfsorientierte Lüftung (accessed: 2020-09-13). https://www.umwelt-campus.de/forschung/projekte/iot-werkstatt/ideen-zur-corona-krise.

[32] hackster.io (user Brian Boyles). Website: Homemade CO2 Sensor Unit (accessed: 2020-09-13). https://www.hackster.io/bfboyles/homemade-co2-sensor-unit-22a9d8.

[33] Arduino Forum (user metropol). Website: Arduino Forum: CO2 level classroom sensor (accessed: 2020-09-13). https://forum.arduino.cc/index.php?topic=702131.

[34] Home Assistant Forum (user Ombra). Website: CO2 monitoring with SCD30 sensor and HA integration (accessed: 2020-09-13). https://community.home-assistant.io/t/co2-monitoring-with-scd30-sensor-and-ha-integration/216708.

[35] Raspberry Pi Forum (user Zentris). Website: In Entwicklung: Co2-Sensor (SCD30) mit ESP8266 und Display SSD1306 (kurze Vorstellung) (accessed: 2020-09-13). https://forum-raspberrypi.de/forum/thread/44717-in-entwicklung-co2-sensor-scd30-mit-esp8266-und-display-ssd1306-kurze-

[36] Kaspar Metz. Coro$_2$sens. https://github.com/kmetz/coro2sens, 2020.

[37] Netzbasteln. Co$_2$narienvogel. https://github.com/netzbasteln/co2narienvogel, 2020.

[38] paulvha. paulvha scd30 library. https://github.com/paulvha/scd30, 2020.

[39] Nikolai Ruhe. Website: Infektometer - Ein CO_2 Messgerät zur Infektionskontrolle in der Schule (accessed: 2020-11-25). http://infektometer.com/.

[40] Guido Burger, Richard Fix, and Klaus-Uwe Gollmer. Der CO2-Warner für die Schule. *Make Magazin*, 2020. https://www.heise.de/select/make/2020/5/2022015381334973804.

[41] Sensirion: The Sensor Company. Datasheet Sensirion SCD30 Sensor Module (Version 0.94). https://www.sensirion.com/fileadmin/user_upload/customers/sensirion/Dokumente/9.5_CO2/Sensirion_CO2_Sensors_SCD30_Datasheet.pdf.

[42] SparkFun Electronics. SparkFun SCD30 CO2 Sensor Library. https://github.com/sparkfun/SparkFun_SCD30_Arduino_Library, 2020.

[43] The Arduino team. Software: The open-source arduino ide. https://www.arduino.cc/en/main/software.

[44] Ventilation The Federation of European Heating and Air Conditioning associations (REHVA). Website: Rehva covid-19 guidance document (version 2020-08-03). hhttps://www.rehva.eu/activities/covid-19-guidance/rehva-covid-19-guidance.

[45] Federal Ministry of Labour and Social Affairs Germany. Website: SARS-CoV-2-Arbeitsschutzregel (version 2020-08-20). https://www.bmas.de/DE/Presse/Meldungen/2020/neue-sars-cov-2-arbeitsschutzregel.html.

[46] National Oceanic and Atmospheric Administration (NOAA). Website: Earth System Research Laboratory (accessed: 2020-09-28). https://www.esrl.noaa.gov/gmd/ccgg/trends/.

[47] NodeMCU. Website: NodeMCU Documentation (accessed: 2020-09-18). https://nodemcu.readthedocs.io/en/master/.

[48] Kate Connolly. Germans embrace fresh air to ward off coronavirus. *The Guardian*, 09 2020.

[49] heise online: Verena Stahmer. Website: CO2-Ampel: Erfahrungsbericht aus der Schule (accessed: 2020-11-25). https://www.heise.de/news/CO2-Ampel-Erfahrungsbericht-aus-der-Schule-4932471.html.

[50] Sensirion: The Sensor Company. Design-In guidelines for SCD30 (Version 0.2). https://www.sensirion.com/fileadmin/user_upload/customers/sensirion/Dokumente/9.5_CO2/Sensirion_CO2_Sensors_SCD30_Assembly_Guideline.pdf.

[51] G. Chiesa, S. Cesari, M. Garcia, M. Issa, and S. Li. Multisensor IoT Platform for Optimising IAQ Levels in Buildings through a Smart Ventilation System. *Sustainability*, 1:5777, 2019, doi:10.3390/su11205777.

[52] Song Chen, T. Yamaguchi, and K. Watanabe. A simple, low-cost non-dispersive infrared CO_2 monitor. *2nd ISA/IEEE Sensors for Industry Conference*, 2003, doi:10.1109/SFICON.2002.1159816.

[53] Chih-Hsiung Shen and Jun-Hong Yeah. Long Term Stable $\Delta - \Sigma$ NDIR Technique Based onTemperature Compensation. *Appl. Sci.*, 9:309, 2019, doi:10.3390/app9020309.

[54] J. Petersen, J. Kristensen, Hagar Elarga, R.K. Andersen, and A. Midtstraum. Accuracy and Air Temperature Dependency of Commercial Low-cost NDIR CO_2 Sensors: An Experimental Investigation. In *Proceedings of COBEE2018*, pages 203–207, 2018.

[55] Sensirion: The Sensor Company. Interface description sensirion scd30 sensor module (version 1.0). https://www.sensirion.com/fileadmin/user_upload/customers/sensirion/Dokumente/9.5_CO2/Sensirion_CO2_Sensors_SCD30_Interface_Description.pdf.

[56] SparkFun Electronic. Website: Installing an Arduino Library. https://learn.sparkfun.com/tutorials/installing-an-arduino-library.

[57] Alf Köhn-Seemann. CO_2 Monitor. https://github.com/alfkoehn/CO2_monitor, 2020.

[58] Adafruit Industries. Adafruit_ssd1306. https://github.com/adafruit/Adafruit_SSD1306, 2020.

[59] johnrickman. Liquidcrystal_i2c. https://github.com/johnrickman/LiquidCrystal_I2C, 2020.

[60] me-no dev. ESPAsyncWebServer. https://github.com/me-no-dev/ESPAsyncWebServer, 2020.

[61] me-no dev. ESPAsyncTCP: Async TCP Library for ESP8266 Arduino. https://github.com/me-no-dev/ESPAsyncTCP, 2020.