

A CO₂ 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

2020-12-10

Abstract

This paper describes the setup of a simple yet reliable CO₂ 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 CO₂ concentration. Measuring the latter can thus help to slow-down the spread of the corona-virus. The program code used for the CO₂ monitor and this documentation is available as a GitHub repository to allow to updates and improvements.

First full version released on 2020-09-22.

License for this document: CC BY-SA 4.0

1 Introduction

It is generally accepted that the CO₂ 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 CO₂ 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). In preliminary studies, it has been recently discovered 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 [6–8]. Note that this seems to happen even if the infected patients show no symptoms of Sars-CoV-2 [9] or not yet any symptoms [10]. It is thus not surprising that the vast majority of Sars-CoV-2 virus transmission seems to happen indoors [11]. With half-life periods of the virus on aerosols on the order of 1 hour [12], 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 [13], a restaurant also from China [14], or from a choir in the US [15]. 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 [16]. In recent recommendations from national authorities, it was suggested to use the CO_2 concentration as an indicator when ventilation is required [17–19].

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 [20].

While commercial CO_2 monitors do exist [21–23], these might be considered too expensive for usage in large quantities in schools or universities and/or have long delivery times at the moment (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 [24].

This work was inspired by a project of the *Hochschule Trier* [25], 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 [26–29]. Furthermore, a small number of projects hosted and maintained as GitHub repositories and webpages using the same CO₂ detector are available [30–33]. We would like to recommend the interested reader in particular to the repository by paulvha [32] as it contains a rather large number of examples and to an article published in the *Make Magazin* [34] which contains a full description including the assembly of a CO₂ monitor similar to the one presented in this paper.

2 The CO₂ monitor

The CO₂ monitor is based on the microelectronic sensor SCD30 which measures the CO₂ concentration and also provides measurements of the ambient temperature and relative humidity [35]. Using Arduino as a programming 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 [36]. Using the Arduino IDE [37], which is available for all major operating systems, the corresponding libraries can be simply included via the library manager.

To make the CO₂ 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. In addition to just showing some numbers, we have included a red LED which lights up as soon as some threshold value of the CO₂ 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)* recommend to issue a warning, corresponding to an orange light, when a value of 800 ppm is reached and prompt to trigger some action like ventilation, corresponding to a red light, when 1000 ppm are reached [38]. The *Federal Ministry of Labour and Social Affairs* of Germany also states a threshold value of 1000 ppm that should not be passed [39]. Note that a value of approximately 410 ppm is the typical CO₂ concentration of air [40].

As controller we decided to use the low-cost open source NodeMCU board [41], as it offers enough flexibility to further extend the functionality of the CO₂ 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 accessed via a web-browser or an app on a smartphone.

A prototype of the CO₂ monitor is shown in Fig. 1. As one can see, it

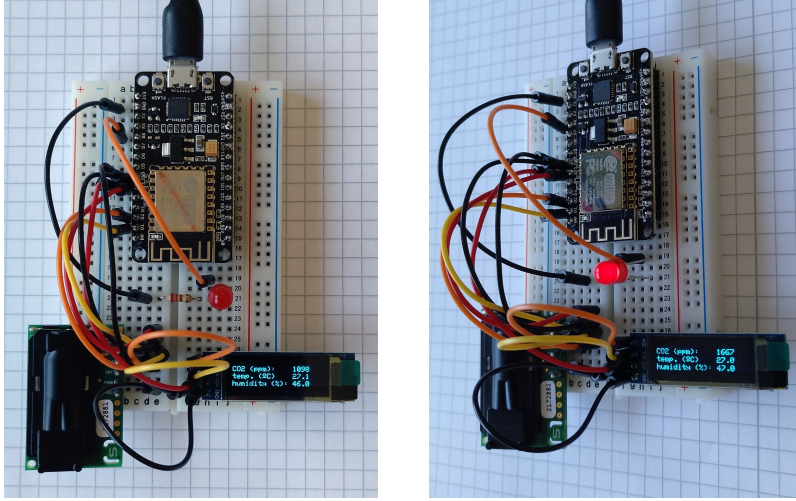


Figure 1: Assembled and working prototype of the CO₂ monitor, (*left*) with a measured CO₂ concentration below the threshold and (*right*) above it (note the red LED).

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 CO₂ 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.

2.1 Positioning in a room

The CO₂ monitor should be placed at a position in a room where it is not exposed to flowing air as this would distort the measured CO₂ 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 ventilation¹. Figure 2 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 CO₂ 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

¹Although cross ventilation is more efficient than impact ventilation ("querlüften" vs. "stoßlüften" [42]) one has to take care that behind the open door in Fig. 2 another open window is needed otherwise one could release part of the classroom air and its potentially infectious aerosols into the corridor which connects different classrooms or lecture halls [43].

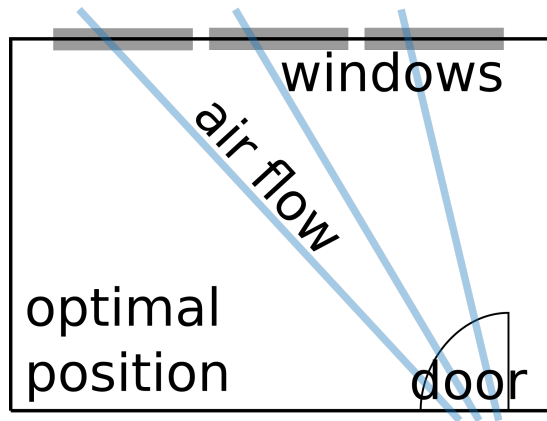


Figure 2: Optimal position of the CO₂ monitor in a room where ventilation by an air flow across the room (cross ventilation) can be applied.

on the CO₂ monitor instead of paying attention to the class or lecture. An easy solution would be to show only CO₂ concentrations round to 100 ppm, for example print “CO₂ in 100 ppm: 8” to a display [43].

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 CO₂ monitor itself.

3 Required parts

The CO₂ 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 CO₂ 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 CO₂ monitor are listed in Table 3. The display can be easily replaced by an OLED of larger size. One could also 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 (signifi-

Element	Quantity	Price
SCD30 (CO ₂ sensor)	1	45 €
NodeMCU EPS8266	1	8 €
0.91" OLED display	1	5 €
red LED	1	0.2 €
220 Omega resistor	1	.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 CO₂ monitor as presented in this paper (note that the prices were obtained in 09/2020 and may vary).

cantly for some of the components) when ordering larger quantities.

For the prototype design of the CO₂ 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 [44], 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 CO₂ sensor

The SCD30 has been chosen because it performs direct measurements of the CO₂ 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 CO₂ 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 CO₂ concentration, as discussed in Sec. 1. For a discussion about monitoring VOC and CO₂ concentration with self-assembled devices we would like to point the interested reader to e.g. Ref. [45].

4.1 Technical specifications

According to the datasheet of the SCD30 [35], the CO₂ 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 CO₂ concentration.

4.2 Nondispersive infrared technique

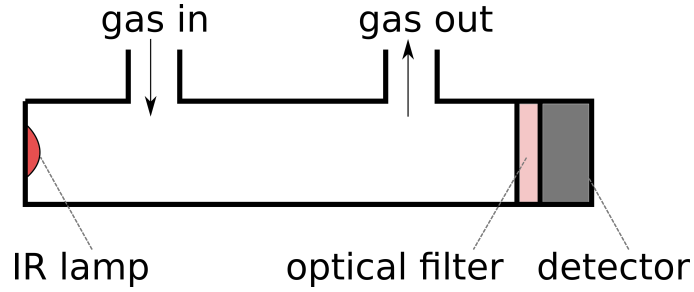


Figure 3: Sketch of a sensor using the nondispersive infrared technique to measure CO₂ concentration.

The CO₂ concentration is measured using the so-called *nondispersive infrared* technique (NDIR). It is the most common sensor type used in industry to measure the CO₂ concentration. Its principle is sketched in Fig. 3. 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 $4.26\ \mu\text{m}$ absorption band of CO₂ 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 $4.26\ \mu\text{m}$ 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 CO₂ molecules in the tube which then allows to calculate the CO₂ concentration using the Beer-Lambert law:

$$I_1 = I_0 e^{-\kappa C l}, \quad (1)$$

where κ is the absorption coefficient of CO₂, 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 [46], 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 [46]. 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 CO₂ 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 CO₂ monitor can be assembled in various ways, here we will restrict ourselves to the case of a simple prototype design on a breadboard as shown in Fig. 4. The connection between the NodeMCU (with the ESP8266) and the SCD30 sensor is as follows:

NodeMCU		SCD30
GND	→	GND
3.3 V	→	VIN
D2/GPI04	→	RX/SDA
D1/GPI05	→	TX/SCL
GND	→	SEL

The NodeMCU then needs to be connected to the OLED display as fol-

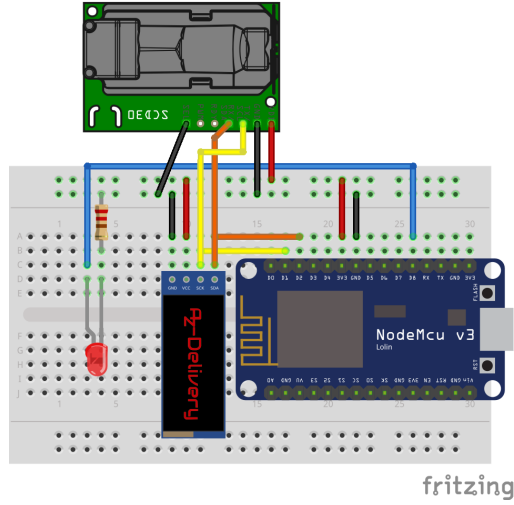


Figure 4: Schematic of a prototype of the CO₂ monitor.

lows:

NodeMCU		OLED display
GND	→	GND
3.3 V	→	VCC
D2/GPI04	→	SDA
D1/GPI05	→	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. 4, 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/GPI015** of the NodeMCU and with its cathode, the shorter leg, via a 220Ω resistor (to limit the current) to ground.

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 [36]. For a tutorial on how to install libraries within the Arduino IDE, see Ref. [47]. 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 CO₂ monitor as described in this paper is available on GitHub [48], 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.

```

1 // -----
2 // Some switches defining general behaviour of the program
3 // -----
4 #define WIFI_ENABLED false           // set to true if WiFi is desired,
5                                     // otherwise corresponding code is not
6                                     // compiled
7 #define DEBUG true                   // activate debugging
8                                     // true: print info + data to serial
9                                     // false: serial monitor is not used
10 #define DISPLAY_OLED                 // OLED display
11 // #define DISPLAY_LCD                 // LCD display
12 // -----

```

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 [49] 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 [50]). 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 [36]).

```

1 // -----
2 // Import all required libraries
3 // -----
4 #include <Wire.h>                     // for I2C communication
5 #ifdef DISPLAY_OLED
6     #include <Adafruit_GFX.h>         // for writing to display
7     #include <Adafruit_SSD1306.h>     // for writing to display
8 #endif
9 #ifdef DISPLAY_LCD
10     #include <LiquidCrystal_I2C.h>
11 #endif
12 #include "SparkFun_SCD30_Arduino_Library.h"
13
14 #include <ESP8266WiFi.h>              // also allows to explicitly turn WiFi
15                                     // off
16 #if WIFI_ENABLED
17     #include <Hash.h>                 // for SHA1 algorithm (for Font Awesome)
18     #include <ESPAsyncTCP.h>
19     #include <ESPAsyncWebServer.h>
20     #include "Webpageindex.h"         // webpage content, same folder as .ino
21                                     // file

```

```

20
21 // Replace with your network credentials
22 const char* ssid      = "ENTER_SSID";
23 const char* password  = "ENTER_PASSWORD";
24 #endif
25 // -----

```

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` [51], based on `ESPAsyncTCP` [52], 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.

```

1 // -----
2 // Hardware configurations and some global constants
3 // -----
4 #define CO2_THRESHOLD1 600
5 #define CO2_THRESHOLD2 1000
6 #define CO2_THRESHOLD3 1500
7
8 #define WARNING_DIODE_PIN D8 // NodeMCU pin for red LED
9
10 #define MEASURE_INTERVAL 10 // seconds, minimum: 2
11
12 #define SCREEN_WIDTH 128 // OLED display width in pixels
13 #define SCREEN_HEIGHT 32 // OLED display height in pixels
14
15 const int lcdColumns = 20; // LCD: number of columns
16 const int lcdRows = 4; // LCD: number of rows
17
18 #define OLED_RESET LED_BUILTIN // OLED reset pin, 4 is default
19 // -1 if sharing Arduino reset pin
20 // using NodeMCU, it is LED_BUILTIN
21
22 const float TempOffset = 5; // temperature offset of the SCD30
23 // 0 is default value
24 // 5 is used in SCD30-library example
25 // 5 also works for most of my devices
26
27 const int altitudeOffset = 300; // altitude of place of operation in
    meters
28 // Stuttgart: approx 300; Uni Stuttgart:
    approx 500
29
30 // update scd30 readings every MEASURE_INTERVAL seconds

```

```

31 const long interval = MEASURE_INTERVAL*1000;
32
33 // use "unsigned long" for variables that hold time
34 // --> value will quickly become too large for an int
35 // store last time scd30 was updated
36 unsigned long previousMilliseconds = 0;
37
38 // switch to perform a forced recalibration
39 // should only be done once in a while and only when outside
40 bool DO_FORCED_RECALIBRATION = false;
41 // -----

```

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.

```

1 // -----
2 // Function prototypes (not needed in Arduino, but good practice imho)
3 // -----
4 void airSensorSetup();
5 void forced_rec calibration();
6 void printToSerial( float co2, float temperature, float humidity);
7 #ifdef DISPLAY_OLED
8     void printToOLED( float co2, float temperature, float humidity);
9     void printEmoji( float value );
10 #endif
11 #ifdef DISPLAY_LCD
12     void printToLCD( float co2, float temperature, float humidity);
13     void scrollLCDText( int row, String message, int delayTime );
14 #endif
15 // -----

```

Listing 4: Function prototypes.

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

```

1 // -----
2 // Hardware declarations
3 // -----
4 #ifdef DISPLAY_OLED
5     // Declaration for an SSD1306 display connected to I2C (SDA, SCL pins)
6     Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
7 #endif
8 #ifdef DISPLAY_LCD
9     // run I2C scanner if LCD address is unknown
10    LiquidCrystal_I2C lcd(0x27, lcdColumns, lcdRows);
11 #endif
12
13 SCD30 airSensor;
14 // -----

```

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.

```

1 #if WIFI_ENABLED
2   // temperature, humidity, CO2 for web-page, updated in loop()
3   float temperature_web = 0.0;
4   float humidity_web    = 0.0;
5   float co2_web         = 0.0;
6
7   // create AsyncWebServer object on port 80 (port 80 for http)
8   AsyncWebServer server(80);
9 #endif

```

Listing 6: Prepare website.

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

```

1 //
2 // PROGMEM stores data in flash memory, default is storing in SRAM
3 // --> usually, only worth to be used for large block of data
4 // R in front of string indicates a RAW string literal
5 // --> no need to escape linebreaks, quotationmarks etc.
6 // --> allows to put full html-website into variable
7 // --> beginning and end of RAW string literal indicated by
8 // =====( ... )=====
9 //
10 const char MAIN_page[] PROGMEM = R"=====(
11 <!doctype html>
12 <html>
13
14 <head>
15   <title>CO2 Monitor</title>
16   <!-- very helpful reference: https://www.w3schools.com -->
17   <meta charset="utf-8">
18   <!-- make webpage fit to your browser, not matter what OS or browser (also
19     adjusts font sizes) -->
20   <meta name="viewport" content="width=device-width, initial-scale=1">
21   <!-- prevent request on favicon, otherwise ESP receives favicon request
22     every time web server is accessed -->
23   <link rel="icon" href="data:,">
24   <!-- load Font Awesome, get integrity and url here: https://fontawesome.
25     com/account/cdn -->
26   <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.14.0/
27     css/all.css" integrity="sha384-HzLeBuhoNPvSl5KYnjx0BT+WB0QEEqLpr0+
28     NBkkek5gbc67FTaL7XIGa2w1LOXbgc" crossorigin="anonymous">
29   <!-- load chart.js library, this could also copied to esp8266 for usage
30     without internet connection -->
31   <script src = "https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.7.3/Chart
32     .min.js"></script>
33   <style>
34     canvas{
35       -moz-user-select: none;
36       -webkit-user-select: none;
37       -ms-user-select: none;
38     }
39     html {
40       font-family      : Arial;
41       display          : inline-block;
42       margin           : 0px auto;
43       text-align       : center;
44     }
45
46     /* data table styling */
47     #dataTable {

```

```

41     font-family      : "Trebuchet MS", Arial, Helvetica, sans-serif;
42     border-collapse  : collapse;
43     width            : 100%;
44 }
45 #dataTable td, #dataTable th {
46     border          : 1px solid #ddd;
47     padding         : 8px;
48 }
49 #dataTable tr:nth-child(even){
50     background-color: #f2f2f2;
51 }
52 #dataTable tr:hover {
53     background-color: #ddd;
54 }
55 #dataTable th {
56     padding-top      : 12px;
57     padding-bottom   : 12px;
58     text-align       : left;
59     background-color : #4CAF50;
60     color            : white;
61 }
62 </style>
63 </head>
64
65 <body>
66     <div style="text-align:center;">
67         <p style="font-size:20px">
68             <b>CO<sub>2</sub> Monitor: data logger (using Chart.js)</b>
69             <button type="button" onclick="downloadData()">Download data</button>
70         </p>
71         <p>
72             <i class="fab fa-github" style="font-size:1.0rem;color:black;"></i>
73             <a href="https://github.com/alfkoehn/CO2_monitor" target="_blank"
74             style="font-size:1.0rem;">Documentation & code on GitHub</a>
75         </p>
76         <i class="fab fa-twitter" style="font-size:1.0rem;color:#1DA1F2;"></i>
77         <span style="font-size:1.0rem;">Contact via twitter: </span>
78         <a href="https://twitter.com/formbar" target="_blank" style="font-size
79         :1.0rem;">&#64;formbar</a>
80     </div>
81     <br>
82     <div class="chart-container" position: relative; height:350px; width:100%"
83     >
84         <canvas id="Chart1" width="400" height="400"></canvas>
85     </div>
86     <br>
87     <div class="chart-container" position: relative; height:350px; width:100%"
88     >
89         <canvas id="Chart2" width="400" height="400"></canvas>
90     </div>
91     <br>
92     <div>
93         <table id="dataTable">
94             <tr>
95                 <th><i class="far fa-clock"></i> Time</th>
96                 <th><i class="fas fa-head-side-cough" style="color:#ffffff;"></i>
97                 CO2 concentration in ppm</th>
98                 <th><i class="fas fa-thermometer-half" style="color:#ffffff;"></i>
99                 Temperature in &deg;C</th>
100                <th><i class="fas fa-tint" style="color:#ffffff;"></i> Humidity in

```

```

100     %</th>
101     </tr>
102 </table>
103 </div>
104 <br>
105 <br>
106 <script>
107 // arrays for data values, will be dynamically filled
108 // if length exceeds threshold, first (oldest) element is deleted
109 var CO2values      = [];
110 var Tvalues        = [];
111 var Hvalues        = [];
112 var timeStamp      = [];
113 var maxArrayLength = 1000;
114
115 // update intervall for getting new data in milliseconds
116 var updateIntervall = 10000;
117
118 // Graphs visit: https://www.chartjs.org
119 // graph for CO2 concentration
120 var ctx = document.getElementById("Chart1").getContext('2d');
121 var Chart1 = new Chart(ctx, {
122     type: 'line',
123     data: {
124         labels: timeStamp, //Bottom Labeling
125         datasets: [{
126             label: "CO2 concentration",
127             fill: 'origin', // 'origin': fill area
128             to x-axis
129             backgroundColor: 'rgba( 243, 18, 156 , .5)', // point fill color
130             borderColor: 'rgba( 243, 18, 156 , 1)', // point stroke color
131             data: CO2values,
132         }],
133     },
134     options: {
135         title: {
136             display: false,
137             text: "CO2 concentration"
138         },
139         maintainAspectRatio: false,
140         elements: {
141             line: {
142                 tension: 0.5 //Smoothening (Curved) of data lines
143             }
144         },
145         scales: {
146             yAxes: [{
147                 display: true,
148                 position: 'left',
149                 ticks: {
150                     beginAtZero: false,
151                     precision: 0,
152                     fontSize: 16
153                 },
154                 scaleLabel: {
155                     display: true,
156                     // unicode for subscript: u+208x,
157                     // for superscript: u+207x
158                     labelString: 'CO\u2082 in ppm',
159                     fontSize: 20

```

```

157     },
158     }]
159 }
160 }
161 });
162 // temperature and humidity graph
163 var ctx2 = document.getElementById("Chart2").getContext('2d');
164 var Chart2 = new Chart(ctx2, {
165     type: 'line',
166     data: {
167         labels: timeStamp, //Bottom Labeling
168         datasets: [{
169             label      : "Temperature",
170             fill        : false,
171             // fill area to xAxis
172             backgroundColor : 'rgba( 243, 156, 18 , 1)', // marker color
173             borderColor    : 'rgba( 243, 156, 18 , 1)', // line Color
174             yAxisID        : 'left',
175             data            : Tvalues,
176         }, {
177             label      : "Humidity",
178             fill        : false,
179             // fill area to xAxis
180             backgroundColor : 'rgba(104, 145, 195, 1)', // marker color
181             borderColor    : 'rgba(104, 145, 195, 1)', // line Color
182             yAxisID        : 'right',
183             data            : Hvalues,
184         }],
185     },
186     options: {
187         title: {
188             display : false,
189             text    : "CO2 Monitor"
190         },
191         maintainAspectRatio: false,
192         elements: {
193             line: {
194                 tension : 0.4 // smoothening (bezier
195                 curve tension)
196             }
197         },
198         scales: {
199             yAxes: [{
200                 id      : 'left',
201                 position : 'left',
202                 scaleLabel: {
203                     display : true,
204                     labelString : 'Temperature in \u00B0C',
205                     fontSize  : 20
206                 },
207                 ticks: {
208                     suggestedMin: 18,
209                     suggestedMax: 30,
210                     fontSize    : 16
211                 }
212             }, {
213                 id      : 'right',
214                 position : 'right',
215                 scaleLabel: {
216                     display : true,
217                     labelString : 'Humidity in %',
218                     fontSize  : 20
219                 },
220                 ticks: {

```



```

218         suggestedMin: 40,
219         suggestedMax: 70,
220         fontSize    : 16
221     }
222 }]]
223 }
224 }
225 });
226
227 // function to dynamically updating graphs
228 // much more efficient than replotting every time
229 function updateCharts() {
230     // update datasets to be plotted
231     Chart1.data.datasets[0].data = CO2values;
232     Chart2.data.datasets[0].data = Tvalues;
233     Chart2.data.datasets[1].data = Hvalues;
234     // update the charts
235     Chart1.update();
236     Chart2.update();
237 };
238
239 // function to download data arrays into csv-file
240 function downloadData() {
241     // build array of strings with data to be saved
242     var data = [];
243     for ( var ii=0 ; ii < CO2values.length ; ii++ ){
244         data.push( [ timeStamp[ii],
245                     Math.round(CO2values[ii]).toString(),
246                     Tvalues[ii].toString(),
247                     Hvalues[ii].toString()
248                 ] );
249     }
250
251     // build String containing all data to be saved (csv-formatted)
252     var csv = 'Time,CO2 in ppm,Temperature in Celsius,Humidity in percent\n';
253     data.forEach(function(row) {
254         csv += row.join(',');
255         csv += "\n";
256     });
257
258     // save csv-string into file
259     // create a hyperlink element (defined by <a> tag)
260     var hiddenElement = document.createElement('a');
261     // similar functions: encodeURIComponent(), encodeURIComponent() (escape() not
262     // recommended)
263     hiddenElement.href = 'data:text/csv;charset=utf-8,'+encodeURIComponent(csv);
264     hiddenElement.target = '_blank';
265     hiddenElement.download= 'CO2monitor.csv';
266     hiddenElement.click();
267 };
268
269 // ajax script to get data repetitively
270 setInterval(function() {
271     // call a function repetitively, intervall set by variable <
272     // updateIntervall>
273     getData();
274 }, updateIntervall);
275
276 function getData() {
277     var xhttp = new XMLHttpRequest();

```

```

277
278 //.onreadystatechange property defines a function to be executed when
279 the readyState changes
280 xmlhttp.onreadystatechange = function() {
281     // "readyState" property holds the status of the "XMLHttpRequest"
282     //     values: 0 --> request not initialized
283     //     1 --> server connection established
284     //     2 --> request received
285     //     3 --> processing request
286     //     4 --> request finished and response is ready
287     // "status" values: 200 --> "OK"
288     //     403 --> "Forbidden"
289     //     404 --> "Page not found"
290     // "this" keyword always refers to objects it belongs to
291     if (this.readyState == 4 && this.status == 200) {
292         //Push the data in array
293
294         // Date() creates a Date object
295         // toLocaleTimeString() returns time portion of Date object as
296         string
297         var time = new Date().toLocaleTimeString();
298
299         // read-only XMLHttpRequest property responseText returns
300         // text received from a server following a request being sent
301         var txt = this.responseText;
302
303         // data from webserver is always a string, parsing with JSON.parse()
304         // to let data beome a JavaScrip object
305         var obj = JSON.parse(txt);
306
307         // add elements to arrays
308         // push() methods adds new items to end of array, returns new length
309         CO2values.push(obj.CO2);
310         Tvalues.push(obj.Temperature);
311         Hvalues.push(obj.Humidity);
312         timeStamp.push(time);
313
314         // if array becomes too long, delete oldest element to not overload
315         graph
316         // also delete first row of data table
317         if (CO2values.length > maxArrayLength) {
318             // shift() method to delete first element
319             CO2values.shift();
320             Tvalues.shift();
321             Hvalues.shift();
322             timeStamp.shift();
323
324             // HTMLTableElement.deleteRow(index), index=-1 for last element
325             document.getElementById("dataTable").deleteRow(-1);
326         }
327
328         // update graphs
329         updateCharts();
330
331         // update data table
332         var table = document.getElementById("dataTable");
333         var row = table.insertRow(1); //Add after headings
334         var cell1 = row.insertCell(0);
335         var cell2 = row.insertCell(1);
336         var cell3 = row.insertCell(2);
337         var cell4 = row.insertCell(3);

```

```

336         cell1.innerHTML = time;
337         cell2.innerHTML = Math.round(obj.CO2);
338         cell3.innerHTML = obj.Temperature;
339         cell4.innerHTML = obj.Humidity;
340     }
341 };
342 xhttp.open("GET", "readData", true); //Handle readData server on ESP8266
343 xhttp.send();
344 }
345
346 </script>
347 </body>
348 </html>
349 )=====";

```

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

```

1 void printToSerial( float co2, float temperature, float humidity) {
2     Serial.print("co2(ppm):");
3     Serial.print(co2, 1);
4     Serial.print(" temp(C):");
5     Serial.print(temperature, 1);
6     Serial.print(" humidity(%):");
7     Serial.print(humidity, 1);
8     Serial.println();
9 }

```

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

```

1 #ifdef DISPLAY_OLED
2 void printToOLED( float co2, float temperature, float humidity) {
3     int
4         x0, x1;           // to align output on OLED display vertically
5
6     x0 = 0;
7     x1 = 84;
8
9     display.clearDisplay();
10    display.setCursor(x0,5);
11    display.print("CO2 (ppm):");
12    display.setCursor(x1,5);
13    // for floats, 2nd parameter in display.print sets number of decimals
14    display.print(co2, 0);
15
16    display.setCursor(x0,15);
17    display.print("temp. ( C)");
18    display.setCursor(x0+7*6,15);
19    display.cp437(true); // enable full 256 char 'Code Page 437' font
20    display.write(248);  // degree symbol
21    display.setCursor(x1,15);
22    display.print(temperature, 1);

```

```

23
24 display.setCursor(x0,25);
25 display.print("humidity (%):");
26 display.setCursor(x1,25);
27 display.print(humidity, 1);
28
29 display.display();
30 }
31 #endif

```

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

```

1 #ifndef DISPLAY_LCD
2 void printToLCD( float co2, float temperature, float humidity) {
3
4     byte degreeSymbol[8] = {
5         0b01100, 0b10010, 0b10010, 0b01100,
6         0b00000, 0b00000, 0b00000, 0b00000
7     };
8     // allocate custom char to a location
9     lcd.createChar(0, degreeSymbol);
10
11     //int waitTime = 2000;
12     lcd.clear();
13     //DrawYoutube();
14     //delay(waitTime);
15
16     // print co2 concentration (1st line, i.e. row 0)
17     lcd.setCursor(0,0);
18     lcd.print("CO2 in ppm");
19     // make output right-aligned
20     lcd.setCursor( (lcdColumns - (int)(log10(co2))+1)), 0);
21     lcd.print(int(round(co2)));
22
23     // print temperature (2nd line, i.e. row 1)
24     lcd.setCursor(0,1);
25     lcd.print("Temp. in ");
26     lcd.write(0);
27     lcd.print("C");
28     // make output right-aligned
29     lcd.setCursor( (lcdColumns - (int)(log10(temperature))+1)), 1);
30     lcd.print(int(round(temperature)));
31
32     // print humidity (3rd line, i.e. row 2)
33     lcd.setCursor(0,2);
34     lcd.print("Humidity in %");
35     // make output right-aligned
36     lcd.setCursor( (lcdColumns - (int)(log10(humidity))+1)), 2);
37     lcd.print(int(round(humidity)));
38
39     //delay(waitTime);
40 }
41 #endif

```

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

Depending on the the CO₂ concentration, the OLED display shows an emoji with the level of happiness being correlated to the value of the CO₂ concentration. The corresponding function to draw that face is given in Listing 11. For the

LCD display, a warning is shown if the CO₂ concentration is too high, the corresponding code is shown in Listing 12.

```

1  #ifndef DISPLAY_OLED
2  void printEmoji( float value ) {
3      // syntax for functions used to draw to OLED:
4      // display.drawBitmap(x, y, bitmap data, bitmap width, bitmap height,
5      // display.drawCircle(x, y, radius, color)
6      // color)
7
8      float start_angle,    // used for smiley mouth
9            end_angle,      // used for smiley mouth
10           i;              // used for smiley mouth
11
12      int smile_x0,
13          smile_y0,
14          smile_r,
15          emoji_r,
16          emoji_x0,
17          emoji_y0,
18          eye_size;
19
20      emoji_r = SCREEN_HEIGHT/4;
21      if (SCREEN_HEIGHT == 32) {
22          emoji_x0 = SCREEN_WIDTH - (1*emoji_r+1);
23          emoji_y0 = emoji_r*3-1;
24          eye_size = 1;
25      } else if (SCREEN_HEIGHT == 64) {
26          emoji_x0 = emoji_r;
27          emoji_y0 = emoji_r*3-1;
28          eye_size = 2;
29      }
30
31      bool plot_all;        // if set, plots all emojis (makes only sense for
32                           // larger oled)
33
34      plot_all = false;
35      if (int(value) == 0) {
36          plot_all = true;
37      }
38
39      if (value < CO2_THRESHOLD1){
40          // very happy smiley face
41
42          display.drawCircle(emoji_x0*1, emoji_y0, emoji_r, WHITE);
43
44          start_angle = 20./180*PI;
45          end_angle = 160./180*PI;
46          smile_r = emoji_r/2;
47          smile_x0 = emoji_x0*1;
48          smile_y0 = emoji_y0+emoji_r/6;
49          for (i = start_angle; i < end_angle; i = i + 0.05) {
50              display.drawPixel(smile_x0 + cos(i) * smile_r, smile_y0 + sin(i) *
51              smile_r, WHITE);
52          }
53
54          display.drawLine(smile_x0+cos(start_angle)*smile_r, smile_y0+sin(
55          start_angle)*smile_r,
56                          smile_x0+cos(end_angle)*smile_r, smile_y0+sin(end_angle
57          )*smile_r,
58                          WHITE);
59      }
60  }

```

```

54 // draw eyes
55 display.fillCircle(emoji_x0*1-emoji_r/2/4*3, smile_y0-emoji_r/3,
56 eye_size, WHITE);
57 display.fillCircle(emoji_x0*1+emoji_r/2/4*3, smile_y0-emoji_r/3,
58 eye_size, WHITE);
59 }
60 if ((value >= CO2_THRESHOLD1 && value < CO2_THRESHOLD2) || (plot_all ==
61 true)) {
62 // happy smiley face
63 if (SCREEN_HEIGHT == 32) {
64 display.drawCircle(emoji_x0, emoji_y0, emoji_r, WHITE);
65 } else if (SCREEN_HEIGHT == 64) {
66 display.drawCircle(emoji_x0 + 2*emoji_r, emoji_y0, emoji_r, WHITE);
67 }
68 // draw mouth
69 if (SCREEN_HEIGHT == 32) {
70 smile_x0 = emoji_x0;
71 } else if (SCREEN_HEIGHT == 64) {
72 smile_x0 = emoji_x0 + 2*emoji_r;
73 }
74 start_angle = 20./180*PI;
75 end_angle = 160./180*PI;
76 smile_r = emoji_r/2;
77 smile_y0 = emoji_y0+emoji_r/6;
78 for (i = start_angle; i < end_angle; i = i + 0.05) {
79 display.drawPixel(smile_x0 + cos(i) * smile_r, smile_y0 + sin(i) *
80 smile_r, WHITE);
81 }
82 // draw eyes
83 display.fillCircle(smile_x0-emoji_r/2/4*3, smile_y0-emoji_r/3, eye_size,
84 WHITE);
85 display.fillCircle(smile_x0+emoji_r/2/4*3, smile_y0-emoji_r/3, eye_size,
86 WHITE);
87 }
88 if ((value >= CO2_THRESHOLD2 && value < CO2_THRESHOLD3) || (plot_all ==
89 true)) {
90 // not so happy smiley face
91 if (SCREEN_HEIGHT == 32) {
92 display.drawCircle(emoji_x0, emoji_y0, emoji_r, WHITE);
93 } else if (SCREEN_HEIGHT == 64) {
94 display.drawCircle(emoji_x0 + 4*emoji_r, emoji_y0, emoji_r, WHITE);
95 }
96 // draw mouth
97 if (SCREEN_HEIGHT == 32) {
98 smile_x0 = emoji_x0;
99 } else if (SCREEN_HEIGHT == 64) {
100 smile_x0 = emoji_x0 + 4*emoji_r;
101 }
102 display.drawLine(smile_x0-emoji_r/2/4*3, emoji_y0+emoji_r/2,
103 smile_x0+emoji_r/2/4*3, emoji_y0+emoji_r/2,
104 WHITE);
105 // draw eyes
106 display.fillCircle(smile_x0-emoji_r/2/4*3, smile_y0-emoji_r/3, eye_size,
107 WHITE);
108 display.fillCircle(smile_x0+emoji_r/2/4*3, smile_y0-emoji_r/3, eye_size,
109 WHITE);

```

```

107 }
108 if ((value >= CO2_THRESHOLD3) || (plot_all == true)) {
109     // sad smiley face
110
111     if (SCREEN_HEIGHT == 32) {
112         display.drawCircle(emoji_x0, emoji_y0, emoji_r, WHITE);
113     } else if (SCREEN_HEIGHT == 64) {
114         display.drawCircle(emoji_x0 + 6*emoji_r-1, emoji_y0, emoji_r, WHITE);
115     }
116
117     // draw mouth
118     if (SCREEN_HEIGHT == 32) {
119         smile_x0 = emoji_x0;
120     } else if (SCREEN_HEIGHT == 64) {
121         smile_x0 = emoji_x0 + 6*emoji_r;
122     }
123     start_angle = 200./180*PI;
124     end_angle = 340./180*PI;
125     smile_r = emoji_r/2;
126     smile_y0 = emoji_y0+emoji_r/6;
127     for (i = start_angle; i < end_angle; i = i + 0.05) {
128         display.drawPixel(smile_x0 + cos(i) * smile_r, smile_y0+emoji_r/2 +
129             sin(i) * smile_r, WHITE);
130     }
131
132     // draw eyes
133     display.fillCircle(smile_x0-emoji_r/2/4*3, smile_y0-emoji_r/3, eye_size,
134         WHITE);
135     display.fillCircle(smile_x0+emoji_r/2/4*3, smile_y0-emoji_r/3, eye_size,
136         WHITE);
137 }
138 display.display();
139 }
140 #endif

```

Listing 11: Function which prints smileys to the OLED display depending on the value of the CO₂ concentration.

```

1 #ifndef DISPLAY_LCD
2 // Parameters:
3 // row:         where text will be printed
4 // message:     text to scroll
5 // delayTime:   time between character shifting
6 // inspired by https://randomnerdtutorials.com/esp32-esp8266-i2c-lcd-arduino-
7 // -ide/
8 void scrollLCDText( int row, String message, int delayTime ){
9     // add whitespaces equal to no. LCD-columns at beginning of string
10    for (int i=0; i<lcdColumns ; ++i) {
11        message = " "+message;
12    }
13    message = message+" ";
14    // emulate the scrolling by printing substrings sequentially
15    for (int pos=0 ; pos<message.length(); ++pos){
16        lcd.setCursor(0,row);
17        lcd.print(message.substring(pos, pos+lcdColumns));
18        delay(delayTime);
19    }
20 }
21 #endif

```

Listing 12: Function which prints a warning to the LCD display depending

on the value of the CO₂ 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.

```
1 void setup(){
2   if (DEBUG == true) {
3     // initialize serial monitor at baud rate of 115200
4     Serial.begin(115200);
5     delay(1000);
6     Serial.println("Using SCD30 to get: CO2 concentration, temperature,
7     humidity");
8   }
9   // initialize I2C
10  Wire.begin();
11
12  // initialize LED pin as an output
13  pinMode(WARNING_DIODE_PIN, OUTPUT);
14
15  #if WIFI_ENABLED
16    /* Explicitly set ESP8266 to be a WiFi-client, otherwise, it would, by
17    default, try to act as both, client and access-point, and could cause
18    network-issues with other WiFi-devices on your WiFi-network. */
19    WiFi.mode(WIFI_STA);
20    WiFi.begin(ssid, password); // connect to Wi-Fi
21    if (DEBUG == true)
22      Serial.println("Connecting to WiFi");
23    while (WiFi.status() != WL_CONNECTED) {
24      delay(1000);
25      if (DEBUG == true)
26        Serial.print(".");
27    }
28    IPAddress ip = WiFi.localIP();
29    if (DEBUG == true)
30      Serial.println(ip);
31
32    // -----
33    // This is executed when you open the IP in browser
34    // -----
35    server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
36      // note: do NOT load MAIN_page into a String variable
37      // this might not work (probably too large)
38      request->send_P(200, "text/html", MAIN_page );
39    });
40
41    // this page is called by java Script AJAX
42    server.on("/readData", HTTP_GET, [](AsyncWebServerRequest *request){
43      // putting all values into one big string
44      // inspiration: https://circuits4you.com/2019/01/11/nodemcu-esp8266-
45      // arduino-json-parsing-example/
46      String data2send = "{\"CO0\": \""+String(co2_web)
47      + "\", \"Temperature\": \""+String(temperature_web)
48      + "\", \"Humidity\": \""+String(humidity_web) + "\"}";
49      request->send_P(200, "text/plain", data2send.c_str());
50    });
51    // -----
```



```

52 server.begin();
53 #else
54 WiFi.mode( WIFI_OFF );           // explicitly turn WiFi off
55 WiFi.forceSleepBegin();          // explicitly turn WiFi off
56 delay( 1 );                      // required to apply WiFi changes
57 if (DEBUG == true)
58     Serial.println("WiFi is turned off.");
59 #endif
60
61 #ifdef DISPLAY_OLED
62 // SSD1306_SWITCHCAPVCC: generate display voltage from 3.3V internally
63 if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { // Address 0x3C for 128
64     x32
65     if (DEBUG == true)
66         Serial.println(F("SSD1306 allocation failed"));
67     for(;;);                      // don't proceed, loop forever
68 }
69 display.display();                // initialize display
70 // library will show Adafruit logo
71 delay(2000);                     // pause for 2 seconds
72 display.clearDisplay();           // clear the buffer
73 display.setTextSize(1);           // has to be set initially
74 display.setTextColor(WHITE);     // has to be set initially
75
76 // move cursor to position and print text there
77 display.setCursor(2,5);
78 display.println("CO2 monitor");
79 display.println("twitter.com/formbar");
80 #if WIFI_ENABLED
81     display.println(ip);
82 #else
83     display.println("WiFi disabled");
84 #endif
85
86 // write previously defined emojis to display
87 if (SCREEN_HEIGHT == 32) {
88     printEmoji(400);
89     delay(2000);
90     printEmoji(600);
91     delay(2000);
92     printEmoji(1200);
93     delay(2000);
94     printEmoji(1800);
95     delay(2000);
96 } else {
97     printEmoji(0);
98 }
99
100 display.display();                // write display buffer to display
101 #endif
102
103 #ifdef DISPLAY_LCD
104 lcd.init();                      // initialize LCD
105 lcd.backlight();                 // turn on LCD backlight
106 lcd.setCursor(0,0);              // set cursor to (column,row)
107 lcd.print("WiFi connected");
108 lcd.setCursor(0,1);
109 lcd.print(ip);
110 delay(2000);
111 #endif
112 // turn warning LED on and off to test it

```

```

113 digitalWrite(WARNING_DIODE_PIN, HIGH);
114 delay(2000*2);
115 digitalWrite(WARNING_DIODE_PIN, LOW);
116
117 // initialize SCD30
118 airSensorSetup();
119 }

```

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.

```

1 void airSensorSetup(){
2
3     bool autoSelfCalibration = false;
4
5     // start sensor using the Wire port, but disable the auto-calibration
6     if (airSensor.begin(Wire, autoSelfCalibration) == false) {
7         if (DEBUG == true)
8             Serial.println("Air sensor not detected. Please check wiring. Freezing
9             ...");
10        while (1)
11            ;
12    }
13
14    // SCD30 has data ready at maximum every two seconds
15    // can be set to 1800 at maximum (30 minutes)
16    //airSensor.setMeasurementInterval(MEASURE_INTERVAL);
17
18    // altitude compensation in meters
19    // alternatively, one could also use:
20    // airSensor.setAmbientPressure(pressure_in_mBar)
21    delay(1000);
22    airSensor.setAltitudeCompensation(altitudeOffset);
23
24    float T_offset = airSensor.getTemperatureOffset();
25    Serial.print("Current temp offset: ");
26    Serial.print(T_offset, 2);
27    Serial.println("C");
28
29    // note: this value also depends on how you installed
30    // the SCD30 in your device
31    airSensor.setTemperatureOffset(TempOffset);
32 }

```

Listing 14: Setup code for the SCD30.

```

1 void forced_rec calibration(){
2     // note: for best results, the sensor has to be run in a stable
3     // environment
4     // in continuous mode at a measurement rate of 2s for at least two
5     // minutes before applying the FRC command and sending the reference
6     // value
7     // quoted from "Interface Description Sensirion SCD30 Sensor Module"
8
9     String counter;
10
11     int CO2_offset_calibration = 410;

```

```

10
11  if (DEBUG == true){
12      Serial.println("Starting to do a forced recalibration in 10 seconds");
13  }
14
15  #ifdef DISPLAY_OLED
16      display.clearDisplay();
17      display.setCursor(0,0);
18      display.println("Warning:");
19      display.println("forced recalibration");
20      display.display();
21
22      for (int ii=0; ii<10; ++ii){
23          counter = String(10-ii);
24          display.setCursor(ii*9,20);
25          display.print(counter);
26          display.display();
27          delay(1000);
28      }
29  #endif
30
31      airSensor.setForcedRecalibrationFactor(CO2_offset_calibration);
32
33  #ifdef DISPLAY_OLED
34      delay(1000);
35      display.clearDisplay();
36      display.setCursor(0,0);
37      display.println("Successfully recalibrated!");
38      display.println("Only required ~once per year");
39      display.display();
40  #endif
41
42      delay(5000);
43  }

```

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 CO₂ 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).

```

1  void loop(){
2
3      float
4          co2_new,
5          temperature_new,
6          humidity_new;
7
8      unsigned long currentMilliseconds;
9
10     // get milliseconds passed since program started to run

```

```

11 currentMilliseconds = millis();
12
13 // forced recalibration requires 2 minutes of stable environment in
14 // advance
15 if ((DO_FORCED_RECALIBRATION == true) && (currentMilliseconds > 120000)) {
16     forced_recalibration();
17     DO_FORCED_RECALIBRATION = false;
18 }
19
20 if (currentMilliseconds - previousMilliseconds >= interval) {
21     // save the last time you updated the DHT values
22     previousMilliseconds = currentMilliseconds;
23
24     if (airSensor.dataAvailable()) {
25         // get updated data from SCD30 sensor
26         co2_new = airSensor.getCO2();
27         temperature_new = airSensor.getTemperature();
28         humidity_new = airSensor.getHumidity();
29
30         // print data to serial console
31         if (DEBUG == true)
32             printToSerial(co2_new, temperature_new, humidity_new);
33
34         // print data to display
35         #ifdef DISPLAY_OLED
36             printToOLED(co2_new, temperature_new, humidity_new);
37             // print smiley with happiness according to CO2 concentration
38             printEmoji( co2_new);
39         #endif
40         #ifdef DISPLAY_LCD
41             printToLCD(co2_new, temperature_new, humidity_new);
42             if (co2_new > CO2_THRESHOLD3)
43                 scrollLCDText( 3, "LUEFTEN", 250 );
44         #endif
45         // if CO2-value is too high, issue a warning
46         if (co2_new >= CO2_THRESHOLD3) {
47             digitalWrite(WARNING_DIODE_PIN, HIGH);
48         } else {
49             digitalWrite(WARNING_DIODE_PIN, LOW);
50         }
51         #if WIFI_ENABLED
52             // updated values for webpage
53             co2_web = co2_new;
54             temperature_web = temperature_new;
55             humidity_web = humidity_new;
56         #endif
57     }
58     delay(100);
59 }

```

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] 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.
- [10] Kaiyuan Sun, Wei Wang, Lidong Gao, Yan Wang, Kaiwei Luo, Lingshuang Ren, Zhifei Zhan, Xinghui Chen, Shanlu Zhao, Yiwei Huang, Qianlai Sun, Ziyang 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.
 - [11] 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.
 - [12] 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.
 - [13] 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.
 - [14] 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.
 - [15] 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.
 - [16] A. Hartmann and M. Kriegel. Risk assessment of aerosols loaded with virus based on co₂-concentration. *Preprint server of the Technische Universität Berlin*, 2020, doi:10.14279/depositonce-10362.
 - [17] 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.

- [18] 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>.
- [19] 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.
- [20] 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.
- [21] NALTIC Industrials, LLC. Aranet4 co₂ meter. <https://naltic.com/aranet4-co2.html#>.
- [22] TFA Dostmann. AIRCO₂NTROL. <https://www.tfa-dostmann.de/en/produkte/professional-usage-16/measuring-instruments-for-professional-usage/co2-measuring-instruments/>.
- [23] Wöhler Technik GmbH. CO₂ Messgerät. <https://www.woehler.de/shop/co2-messgeraet.html>.
- [24] B. Watzka and R. Girwidz. Ist die Luft zu schlecht zum Lernen? Nicht-dispersive IR-CO₂-Gassensoren im Physikunterricht. *Physik und Didaktik in Schule und Hochschule*, 10:22–33, 2011.
- [25] 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>.
- [26] hackster.io (user Brian Boyles). Website: Homemade CO₂ Sensor Unit (accessed: 2020-09-13). <https://www.hackster.io/bfboyles/homemade-co2-sensor-unit-22a9d8>.

- [27] Arduino Forum (user metropol). Website: Arduino Forum: CO2 level classroom sensor (accessed: 2020-09-13). <https://forum.arduino.cc/index.php?topic=702131>.
- [28] 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>.
- [29] 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>
- [30] Kaspar Metz. Coro₂sens. <https://github.com/kmetz/coro2sens>, 2020.
- [31] Netzbasteln. Co₂narienvogel. <https://github.com/netzbasteln/co2narienvogel>, 2020.
- [32] paulvha. paulvha scd30 library. <https://github.com/paulvha/scd30>, 2020.
- [33] Nikolai Ruhe. Website: Infektometer - Ein CO₂ Messgerät zur Infektionsskontrolle in der Schule (accessed: 2020-11-25). <http://infektometer.com/>.
- [34] 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>.
- [35] 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.
- [36] SparkFun Electronics. SparkFun SCD30 CO2 Sensor Library. https://github.com/sparkfun/SparkFun_SCD30_Arduino_Library, 2020.
- [37] The Arduino team. Software: The open-source arduino ide. <https://www.arduino.cc/en/main/software>.

- [38] Ventilation The Federation of European Heating and Air Conditioning associations (REHVA). Website: Rehva covid-19 guidance document (version 2020-08-03). <https://www.rehva.eu/activities/covid-19-guidance/rehva-covid-19-guidance>.
- [39] 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>.
- [40] National Oceanic and Atmospheric Administration (NOAA). Website: Earth System Research Laboratory (accessed: 2020-09-28). <https://www.esrl.noaa.gov/gmd/ccgg/trends/>.
- [41] NodeMCU. Website: NodeMCU Documentation (accessed: 2020-09-18). <https://nodemcu.readthedocs.io/en/master/>.
- [42] Kate Connolly. Germans embrace fresh air to ward off coronavirus. *The Guardian*, 09 2020.
- [43] 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>.
- [44] 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.
- [45] 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.
- [46] 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.
- [47] SparkFun Electronic. Website: Installing an Arduino Library. <https://learn.sparkfun.com/tutorials/installing-an-arduino-library>.
- [48] Alf Köhn-Seemann. CO₂ Monitor. https://github.com/alfkoehn/CO2_monitor, 2020.

- [49] Adafruit Industries. Adafruit_ssd1306. https://github.com/adafruit/Adafruit_SSD1306, 2020.
- [50] johnrickman. Liquidcrystal_i2c. https://github.com/johnrickman/LiquidCrystal_I2C, 2020.
- [51] me-no dev. ESPAsyncWebServer. <https://github.com/me-no-dev/ESPAsyncWebServer>, 2020.
- [52] me-no dev. ESPAsyncTCP: Async TCP Library for ESP8266 Arduino. <https://github.com/me-no-dev/ESPAsyncTCP>, 2020.