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

2020-11-09

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 to 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). 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 it the infected patients show no symptoms of Sars-CoV-2 [9]. It is thus not surprising that the vast majority of Sars-CoV-2 virus transmission seems to happen indoors [10]. With half-life periods of the virus on aerosols on the order of 1 hour [11], 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. 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 [12]. In recent recommendations from national authorities, it was suggested to use the CO_2 concentration as an indicator when ventilation is required [13–15].

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 [16].

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 example 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 [17].

This work was inspired by a project of the *Hochschule Trier* [18], 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 [19–22]. Furthermore, a small number of GitHub repositories using the same CO_2 detector are available [23–25] (we would like to recommend the interested reader in particular to the repository by paulvha [25] as it contains a rather large number of examples).

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 [26]. 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 [27]. Using the Arduino IDE [28], 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 very 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 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)* 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 [29]. The *Federal Ministry of Labour and Social Affairs* of Germany also states a threshold value of 1000 ppm that should not be passed [30]. Note that a value of approximately 410 ppm is the typical CO_2 concentration of air [31].

As controller we decided to use the low-cost open source NodeMCU board [32], 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 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.

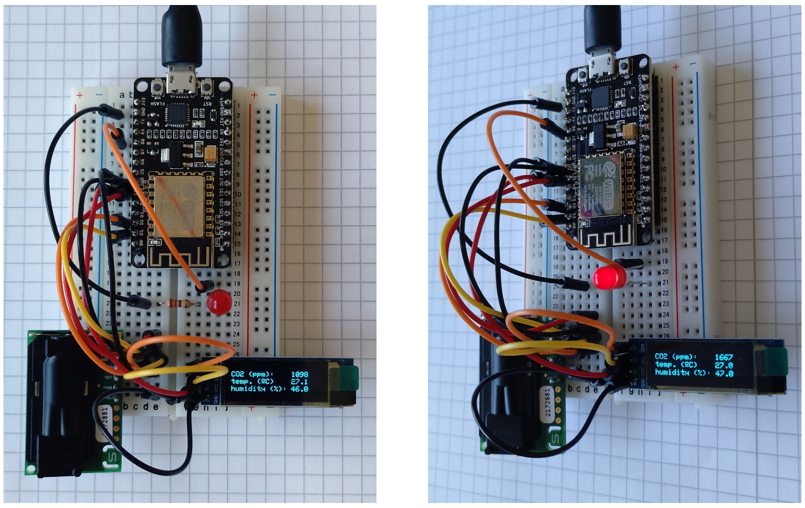

Figure 1: Assembled and working prototype of the CO_2 monitor, *(left)* with a measured CO_2 concentration below the threshold and *(right)* above it (note the red LED).

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 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

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 CO_2 monitor as presented in this paper (note that the prices were obtained in 09/2020 and may vary).

described in a manufacturer's document [33], 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. [34].

4.1 Technical specifications

According to the datasheet of the SCD30 [26], 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

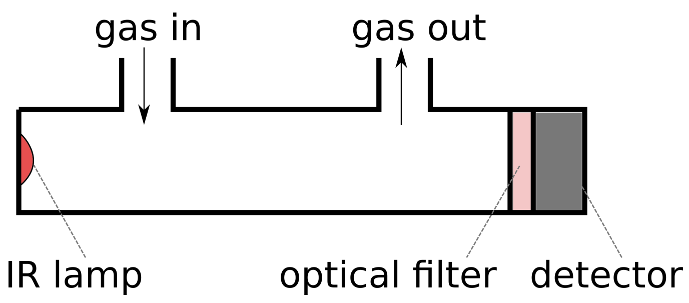

Figure 2: Sketch of a sensor using the nondispersive infrared technique to measure CO_2 concentration.

The CO_2 concentration is measured using the so-called *nondispersive infrared* technique (NDIR). It is the most common sensor type used in industry to measure the CO_2 concentration. Its principle is sketched in Fig. 2. A light source 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 μm 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 μm to pass. A detector then collects the remaining light. The difference between the amount of light emitted by the source and received by the detector is due to the CO_2 molecules in the tube which then allows to calculate the CO_2 concentration.

Using folded optics, i.e. waveguides, for the tube and diodes for the infrared source and detector, allows for a very compact size of the overall sensor on the order of just a few centimeters.

5 Assembly

The CO_2 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. 3. The connection between the NodeMCU (with the ESP8266) and

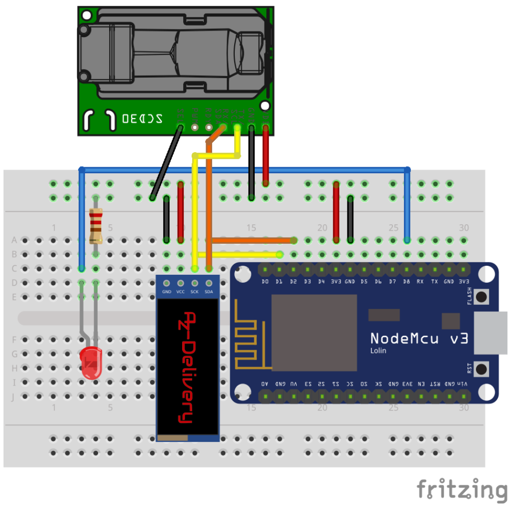

Figure 3: Schematic of a prototype of the CO_2 monitor.

the SCD30 sensor is as follows:

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 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. 3, 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.

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 [27]. For a tutorial on how to install libraries within the Arduino IDE, see Ref. [35]. 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 [36], 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. The `include` statements and some configurations of the program are listed in Listing 1. The `Adafruit_GFX.h` and `Adafruit_SSD1306.h` libraries are used for the OLED display and are required to be installed via the library manager of the Arduino IDE beforehand (alternatively, they are also available on GitHub [37] for manual installation). Note that the display size in pixels 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 [27]).

```
// ---------------------------------------------------------------------
// 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
#define DEBUG true               // activate debugging
                                 // true:  print info + data to serial
    monitor
                                 // false: serial monitor is not used


// ---------------------------------------------------------------------
// Import all required libraries
// ---------------------------------------------------------------------
#include <Wire.h>                  // for I2C communication
#include <Adafruit_GFX.h>          // for writing to display
#include <Adafruit_SSD1306.h>      // for writing to display
#include "SparkFun_SCD30_Arduino_Library.h"

// set to true if WiFi is desired, otherwise corresponding code is not
    compiled
#define WIFI_ENABLED true

#if WIFI_ENABLED
  #include <ESP8266WiFi.h>
  #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

  // Replace with your network credentials
  const char* ssid      = "ENTER_SSID";
  const char* password  = "ENTER_PASSWORD";
#endif

// activate debugging
//   true:  print info + data to serial monitor
//   false: serial monitor is not used
#define DEBUG true

#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

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

// Declaration for an SSD1306 display connected to I2C (SDA, SCL pins)
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

SCD30 airSensor;
// use "unsigned long" for variables that hold time
//  --> value will quickly become too large for an int
unsigned long previousMilliseconds = 0;     // store last time scd30 was
    updated

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

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

Listing 1: Load required libraries and set some configurations.

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` [38], based on `ESPAsyncTCP` [39], 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.

To display the values measured by the SCD30 sensor on a website, we use global variables in the code, as shown in Listing 2. The complete html code for the website is loaded via including it in as a library and then copying into a string variable, called `webpage`.

```
#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;

  // create AsyncWebServer object on port 80 (port 80 for http)
  AsyncWebServer server(80);

  // read html into string
  String webpage = index_html;

  // function for replacing placeholder on webpage with SCD30 values
  String processor(const String& var){
    //Serial.println(var);
    if(var == "CO2"){
      return String(co2_web);
    }
    else if(var == "TEMPERATURE"){
      return String(temperature_web);
    }
    else if(var == "HUMIDITY"){
      return String(humidity_web);
    }
    return String();
  }
#endif
```

Listing 2: Prepare website.

The code for the webpage is shown in Listing 3.

```
const char index_html[] PROGMEM = R"rawliteral(
<!DOCTYPE HTML><html>
<head>
  <!-- make webpage fit to your browser, not matter what OS or browser -->
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <!-- 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">
  <!-- add some CSS style: font, size for header (h2) and paragraph (p) -->
  <!--                    size and format for labels to read          -->
  <style>
    html {
     font-family: Arial;
     display:     inline-block;
     margin:      0px auto;
     text-align:  center;
    }
    h2 {
      font-size:  3.0rem;
    }
    p {
      font-size:  3.0rem;
    }
    .units {
      font-size:  1.2rem;
    }
    .scd30-labels{
      font-size:      1.5rem;
      vertical-align: middle;
      padding-bottom: 15px;
    }
  </style>
</head>
<body>
  <h2>CO2 monitor</h2>
  <!-- paragraph for CO2 concentration -->
  <p>
    <i class="fas fa-head-side-cough" style="color:#ff6600;"></i>
    <span class="scd30-labels">CO2 concentration</span>
    <span id="co2">%CO2%</span>
    <sup class="units">ppm</sup>
  </p>
  <!-- paragraph for temperature -->
  <p>
    <i class="fas fa-thermometer-half" style="color:#059e8a;"></i>
    <span class="scd30-labels">Temperature</span>
    <span id="temperature">%TEMPERATURE%</span>
    <sup class="units">&#8451</sup>
  </p>
  <!-- paragraph for humidity -->
  <p>
    <i class="fas fa-tint" style="color:#00add6;"></i>
    <span class="scd30-labels">Humidity</span>
    <span id="humidity">%HUMIDITY%</span>
    <sup class="units">%</sup>
  </p>
  <!-- paragraph for getting additional information (github) and getting into contact via twitter -->
  <p>
    <i class="fab fa-github" style="font-size:1.0rem;color:black;"></i>
    <span style="font-size:1.0rem;">The CO2 monitor on </span>
    <a href="https://github.com/alfkoehn/CO2_monitor" target="_blank" style="font-size:1.0rem;">GitHub (documentation + code)</a>
  </P>
  <p>
    <i class="fab fa-twitter" style="font-size:1.0rem;color:#1DA1F2;"></i>
    <span style="font-size:1.0rem;">Twitter: </span>
    <a href="https://twitter.com/formbar" target="_blank" style="font-size:1.0rem;">&#64;formbar</a>
  </P>
</body>
<!-- JavaScript to update CO2, temperature and humidity automatically -->
<script>
setInterval(function ( ) {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("co2").innerHTML = this.responseText;
    }
  };
  xhttp.open("GET", "/co2", true);
  xhttp.send();
}, 10000 ) ;

setInterval(function ( ) {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("temperature").innerHTML = this.responseText;
    }
  };
  xhttp.open("GET", "/temperature", true);
  xhttp.send();
}, 10000 ) ;

setInterval(function ( ) {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("humidity").innerHTML = this.responseText;
    }
  };
  xhttp.open("GET", "/humidity", true);
  xhttp.send();
}, 10000 ) ;
</script>
</html>)rawliteral";
```

Listing 3: Code for the webpage.

The function to print the data obtained from the SCD30 to the serial console is shown in Listing 4, the function to print the data to the OLED display is given in Listing 6.

```
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 4: Function which prints data to the serial console.

```
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.write(167);
  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();
}
```

Listing 5: Function which prints data to the OLED display.

Depending on the the CO_2 concentration, the OLED display shows an emoji with the level of happiness being correlated to the CO_2 concentration. The corresponding function to draw that face is given in Listing 6.

```
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;

  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();
}
```

Listing 6: Function which prints smilies to the OLED display depending on the CO_2 concentration.

Listing 8 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);
    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
  // Connect to Wi-Fi
  WiFi.begin(ssid, password);
  if (DEBUG == true)
    Serial.println("Connecting to WiFi");
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    if (DEBUG == true)
      Serial.print(".");
  }
  if (DEBUG == true)
    Serial.println(WiFi.localIP());
#endif

  // SSD1306_SWITCHCAPVCC: generate display voltage from 3.3V internally
  if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { // Address 0x3C for 128
    x32
    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(WiFi.localIP());
#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

  // 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();

#if WIFI_ENABLED
  // Route for root / web page
  server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send_P(200, "text/html", index_html, processor);
  });
  server.on("/co2", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send_P(200, "text/plain", String(co2_web).c_str());
  });
  server.on("/temperature", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send_P(200, "text/plain", String(temperature_web).c_str());
  });
  server.on("/humidity", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send_P(200, "text/plain", String(humidity_web).c_str());
  });
  // Start server
  server.begin();
#endif
}
```

Listing 7: Setup code.

The calibration and setup of the SCD30 is put into a separate function, shown in Listing 8. An additional function, given in Listing 9, performs a forced recalibration of the SCD30.

```
void airSensorSetup(){

  bool autoSelfCalibration = false;

  // altitude of place of operation in meters
  // Stuttgart: approx 300; Uni Stuttgart: approx 500
  int alt = 300;

  // Start sensor using the Wire port, but disable the auto-calibration
  if (airSensor.begin(Wire, autoSelfCalibration) == false) {
  //if (airSensor.begin() == false) {
    if (DEBUG == true)
      Serial.println("Air sensor not detected. Please check wiring. Freezing
    ...");
    while (1)
      ;
  }

  // SCD30 has data ready at maximum every two seconds
  // can be set to 1800 at maximum (30 minutes)
  //airSensor.setMeasurementInterval(2);

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


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

  // note that this value also depends on how you installed the SCD30
  airSensor.setTemperatureOffset(7); //5 is used in example (0 is default
    value)
}
```

Listing 8: 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");
  }

  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);
  }

  airSensor.setForcedRecalibrationFactor(CO2_offset_calibration);

  delay(1000);
  display.clearDisplay();
  display.setCursor(0,0);
  display.println("Successfully recalibrated!");
  display.println("Only required ~once per year");
  display.display();
  delay(5000);

}
```

Listing 9: Function to perform a forced recalibration of the SCD30.

The main code, the `loop` function, is given in Listing 10. 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 the OLED display. Listings 4 and 6 show the code for the two latter functions. 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 (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 (airSensor.dataAvailable()) {
      // get 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 OLED display
      printToOLED(co2_new, temperature_new, humidity_new);

#if WIFI_ENABLED
      // updated values for webpage
      co2_web         = co2_new;
      temperature_web = temperature_new;
      humidity_web    = humidity_new;
#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);
    }

    // print smiley with happiness according to CO2 concentration
    printEmoji( co2_new);
  }
}
```

Listing 10: 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 covid19. *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] 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.

[11] 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.

[12] 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.

[13] A. Vo, A. Gritzki, and K. Bux. Infektionsschutzgerechtes Lften Hinweise und Manahmen in Zeiten der SARS-CoV-2-Epidemie. *Bundesanstalt fr Arbeitsschutz und Arbeitsmedizin (BAuA)*, 2020, doi:10.21934/baua:fokus20200918.

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

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

[16] 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.

[17] 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.

[18] 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`.

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

[20] Arduino Forum (user metropol). Website: Arduino forum: Co2 level classroom sensor (accessed: 2020-09-13). `https://forum.arduino.cc/index.php?topic=702131`.

[21] 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`.

[22] 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-`

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

[24] Netzbasteln. Co_2narienvogel. https://github.com/netzbasteln/co2narienvogel, 2020.

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

[26] 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.

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

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

[29] 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.

[30] 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.

[31] National Oceanic and Atmospheric Administration (NOAA). Website: Earth system research laboratory (accessed: 2020-09-28). https://www.esrl.noaa.gov/gmd/ccgg/trends/.

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

[33] 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.

[34] 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.

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

[36] Alf Köhn-Seemann. Co_2 monitor. https://github.com/alfkoehn/CO2_monitor, 2020.

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

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

[39] me-no dev. Espasynctcp: Async tcp library for esp8266 arduino. https://github.com/me-no-dev/ESPAsyncTCP, 2020.