

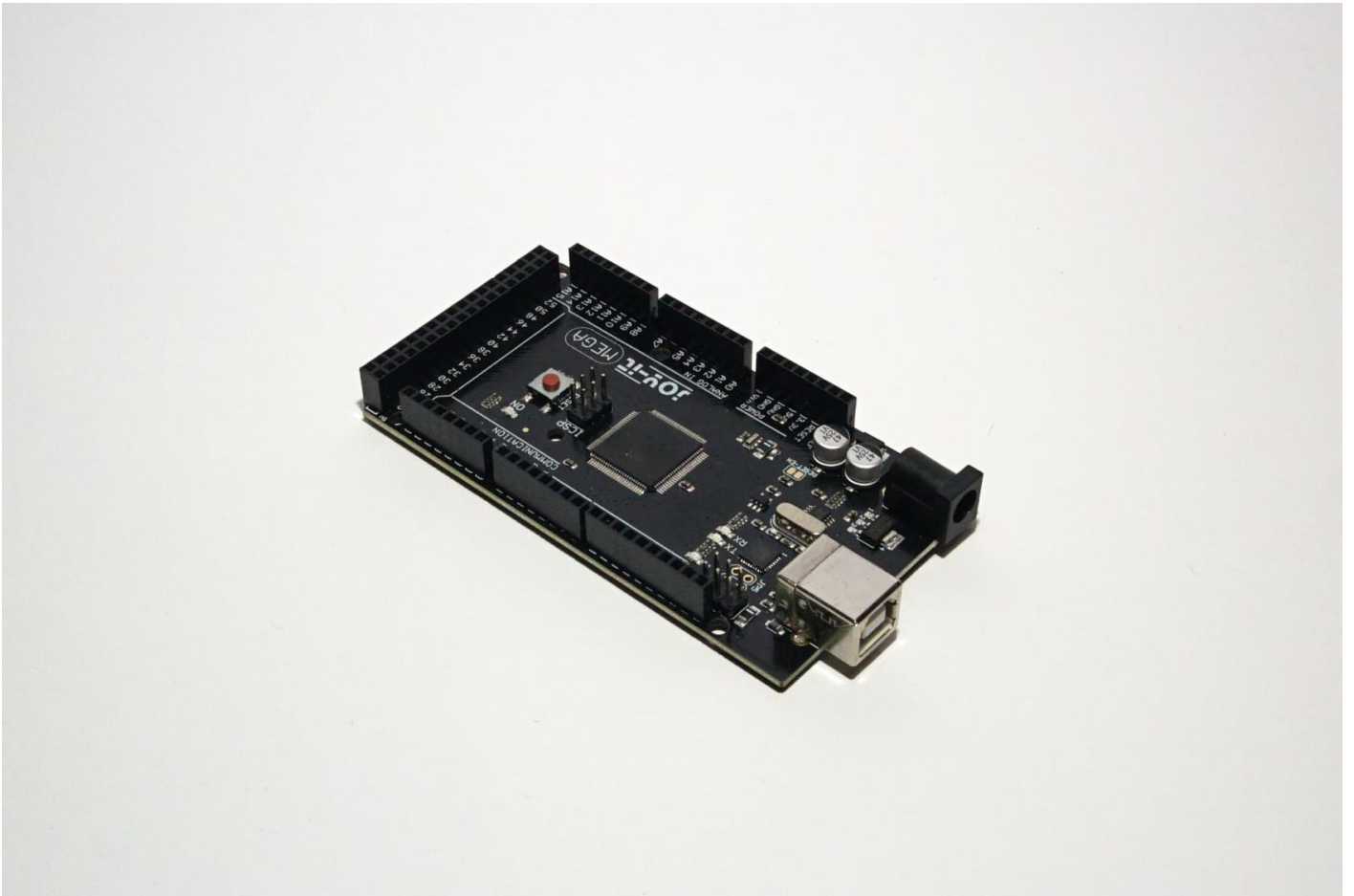
CO2 Melder

Wir bauen einen einfachen CO2 Melder der den CO2 Gehalt der Luft misst und ein Signal ausgibt, sobald bestimmte Werte überschritten werden.

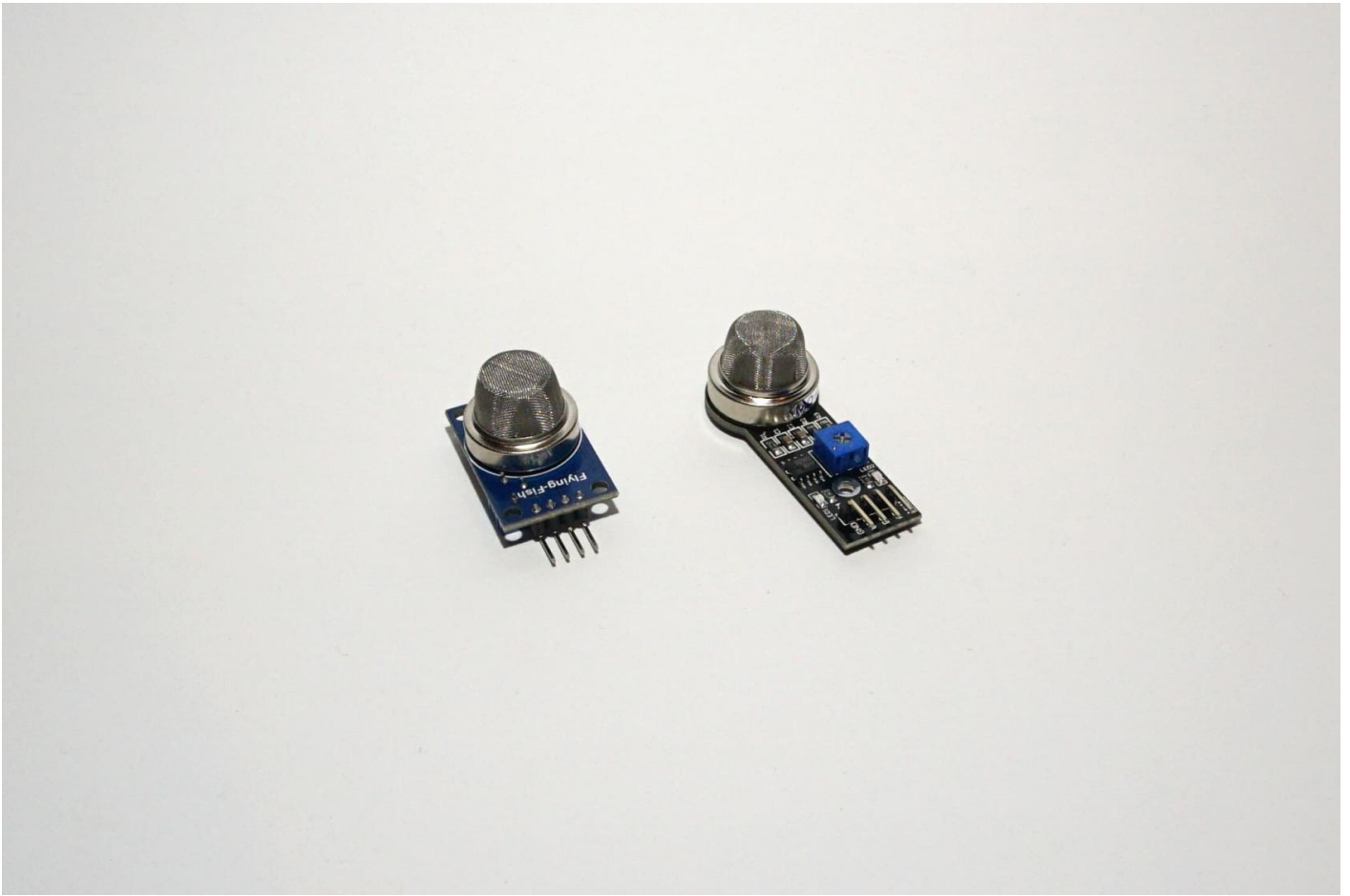
Nachfolgend findet ihr eine Anleitung, sowie alle benötigten Teile. Viel Spaß beim Nachbauen!

Bauteile

- Arduino (egal ob Nano, Uno oder Mega) (<https://amzn.to/32nR6D1>*)
- MQ135 Luftqualität-Sensor (<https://amzn.to/32lIFbz>*)
- Leuchtdioden (im Idealfall 5mm LEDs) (<https://amzn.to/3twDSje>*)
 - rot
 - gelb
 - grün
- Summer (<https://amzn.to/3ed9bcu>*)
- Platine (Rohling: <https://amzn.to/3sAwGSc>* oder <https://amzn.to/2QF6PeA>*)
- Gehäuse (<https://www.prusaprinters.org/de/prints/64081-co2-detector>)
- DIN912 M2x12 (<https://amzn.to/3ttuHAK>*)

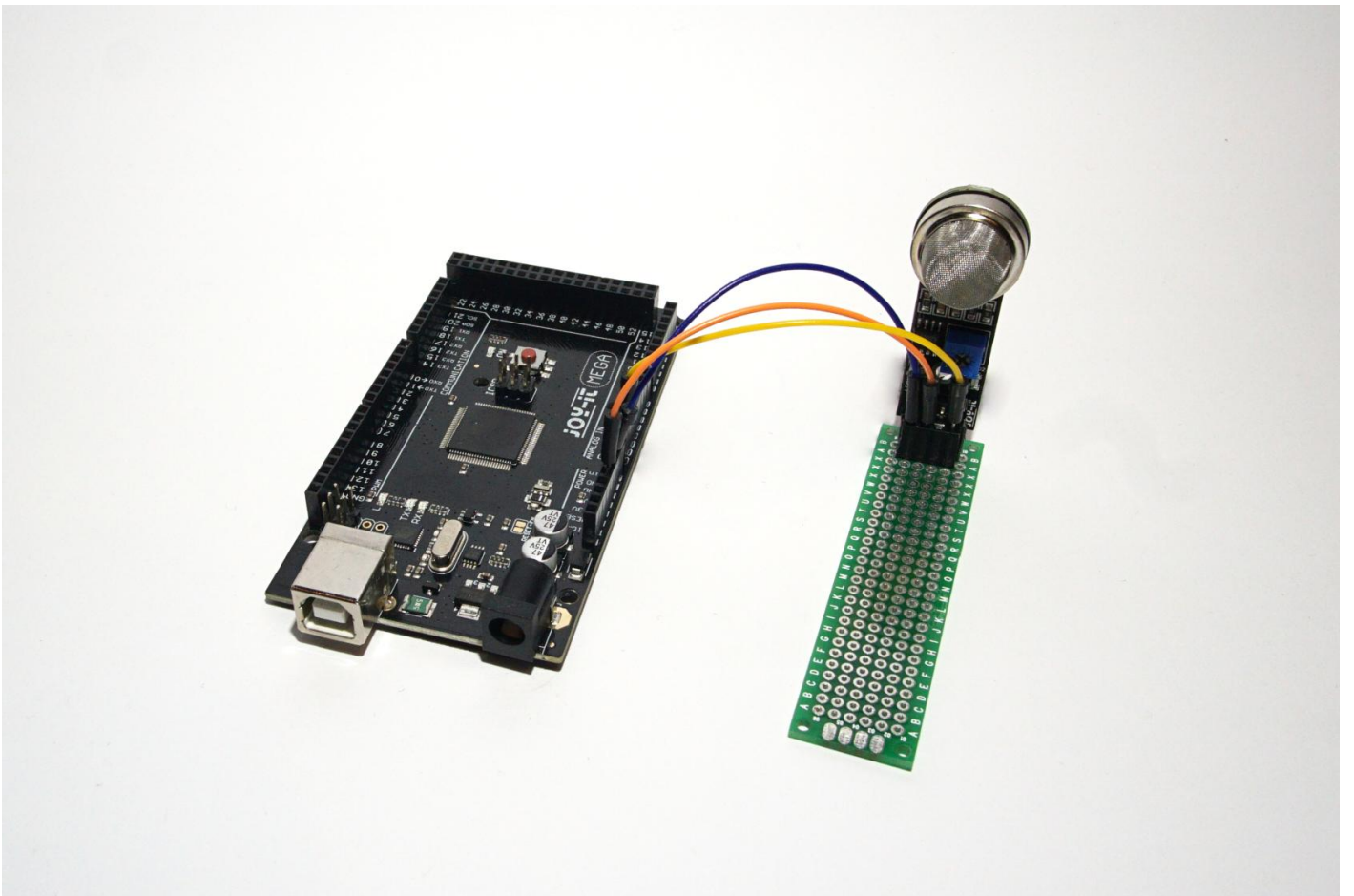


Wir verwenden zum Testen einen Arduino Mega

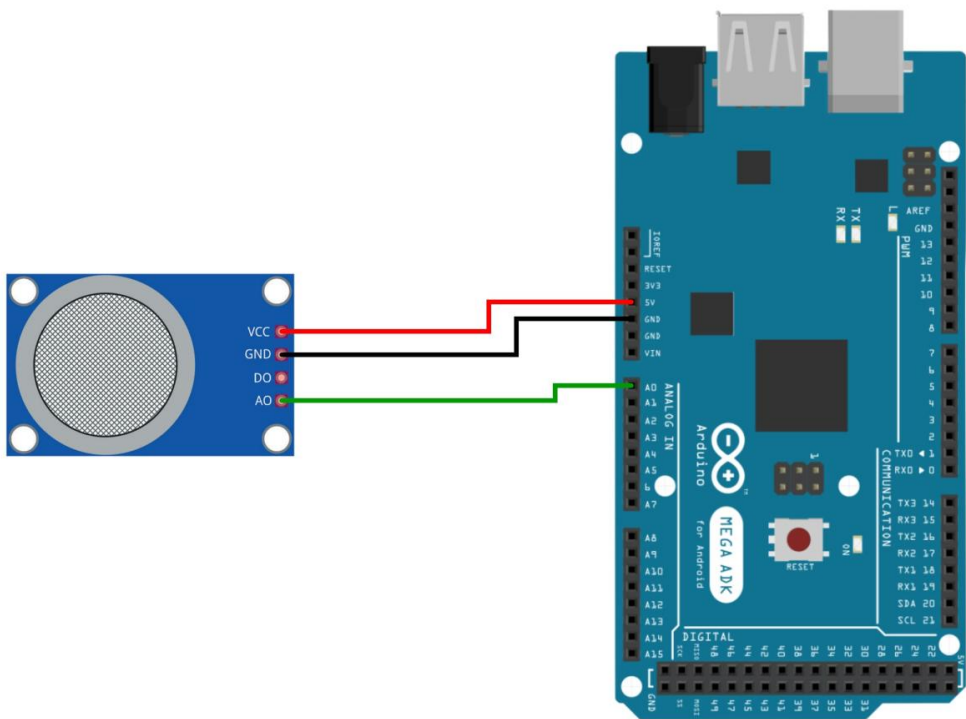


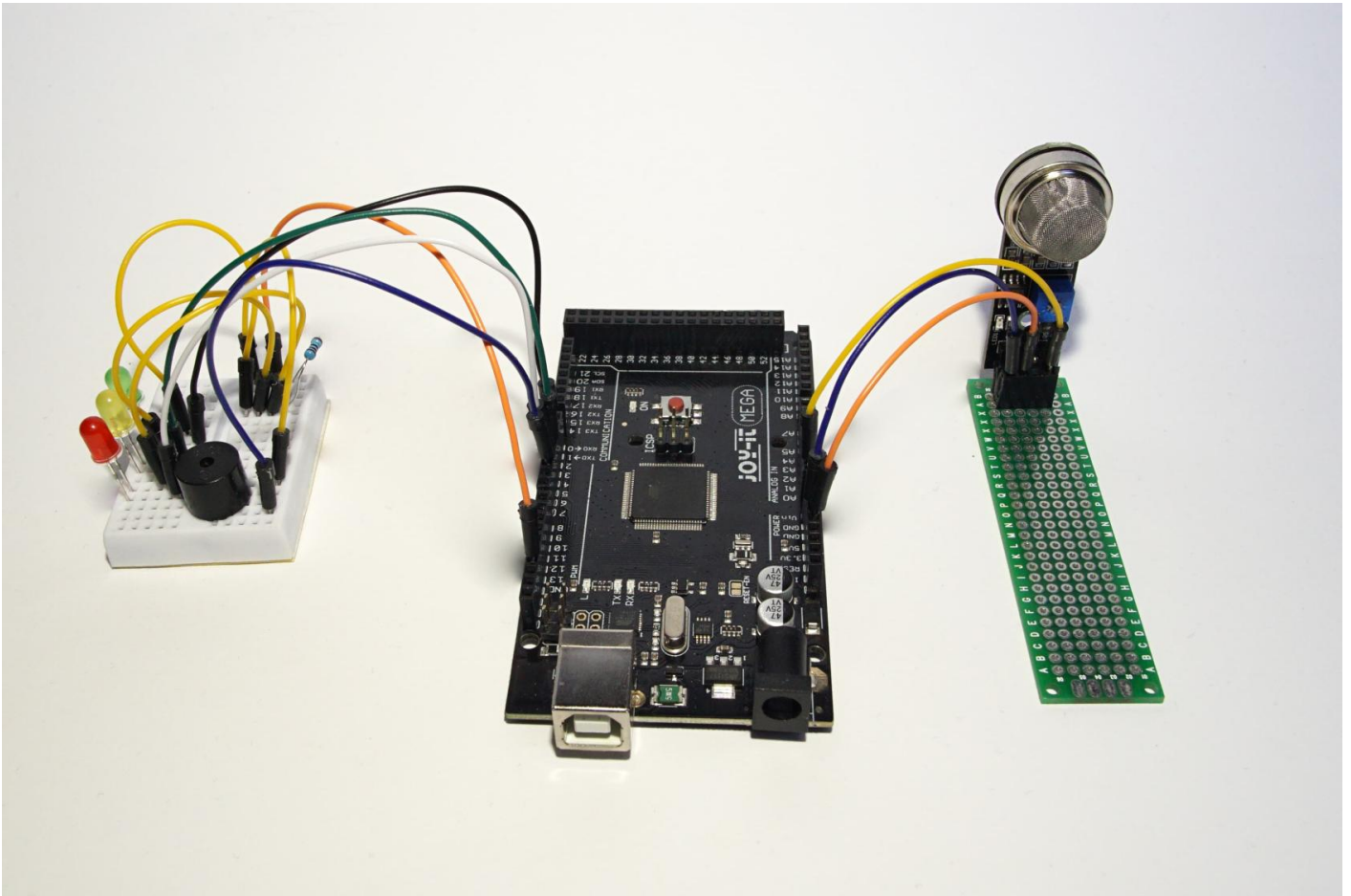
Der Luftqualität-Sensor MQ135 - die Version spielt keine Rolle

Aufbau



Der MQ135 wird an 5V, Masse und einen analog Eingang angeschlossen



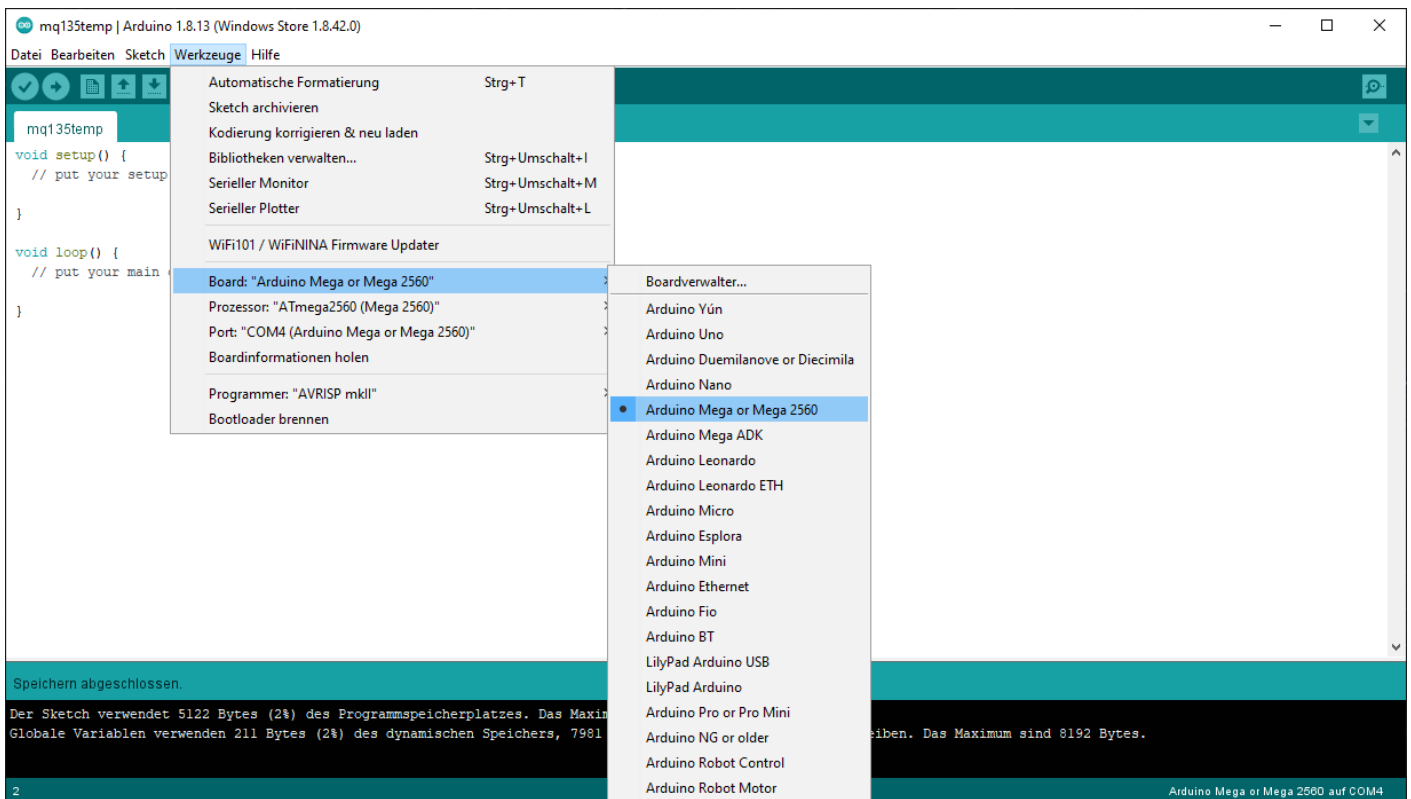


Am CO2 Melder werden drei LEDs installiert, eine grüne, eine gelbe und eine rote. Diese zeigen drei Luftqualitätslevel an. Die grüne leuchtet wenn alles in Ordnung ist, die gelbe soll leuchten, sobald gelüftet werden sollte und die rote soll angehen, wenn der CO2 Gehalt zu hoch wird und gelüftet werden muss. Zusätzlich zu den LEDs wird ein Summer verbaut, der anfängt zu piepsen, sobald die rote LED leuchtet.

Software

Um die Software für unseren CO2 Melder zu schreiben nutzen wir die [Arduino IDE](#)

Nachdem wir die IDE installiert haben, müssen wir das Board umstellen, das wir nutzen. In unserem Fall ist das ein Arduino MEGA Board.

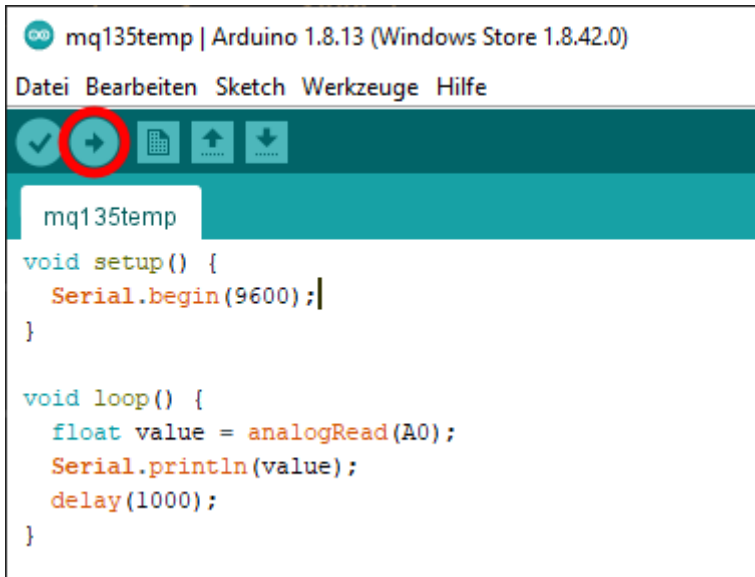


Einstieg

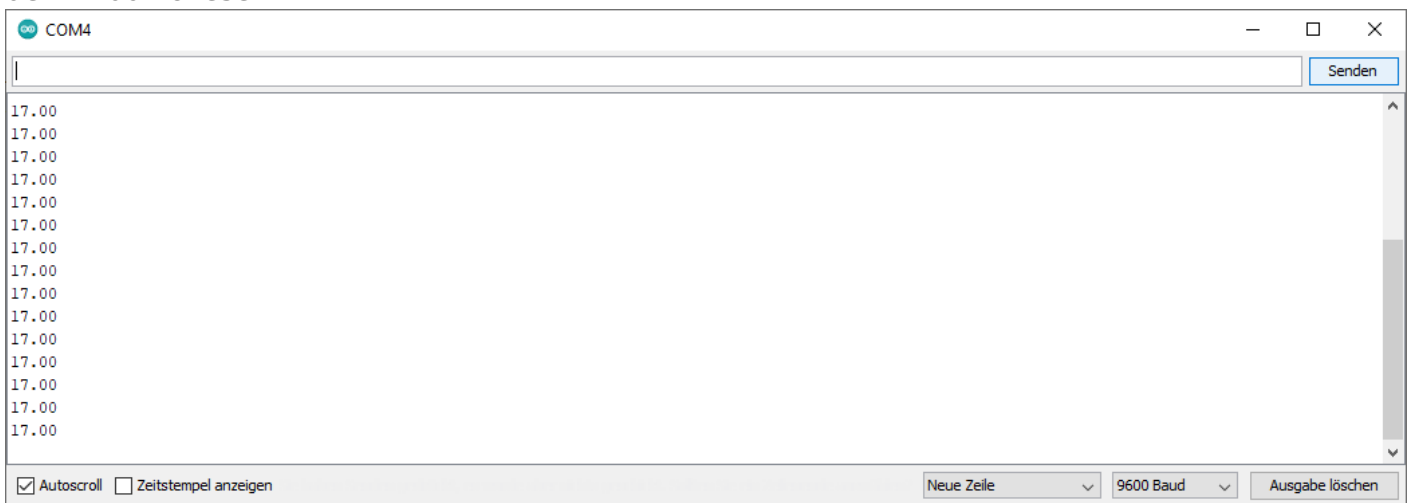
Wir beginnen mit einem einfachen Programm, das die rohen Daten von unserem Sensor liest und über die serielle Schnittstelle ausgibt:

```
void setup() {  
    // setzt die Datenrate für die serielle Datenübertragung  
    Serial.begin(9600);  
}  
  
void loop() {  
    // liest den analolgen Wert an A0  
    float value = analogRead(A0);  
    // schreibt den Wert über die serielle Schnittstelle  
    Serial.println(value);  
    // wartet 1000ms = 1s  
    delay(1000);  
}
```

Wir stellen sicher, dass das Arduino Board an unserem PC über USB angeschlossen ist und unter **Werkzeuge > Port**, der korrekte USB Eingang ausgewählt ist. Dann können wir unser Programm auf den Arduino übertragen.



Über den seriellen Monitor unter **Werkzeuge > Serieller Monitor** können wir die Nachrichten von dem Arduino lesen.



Werte in ppm auslesen

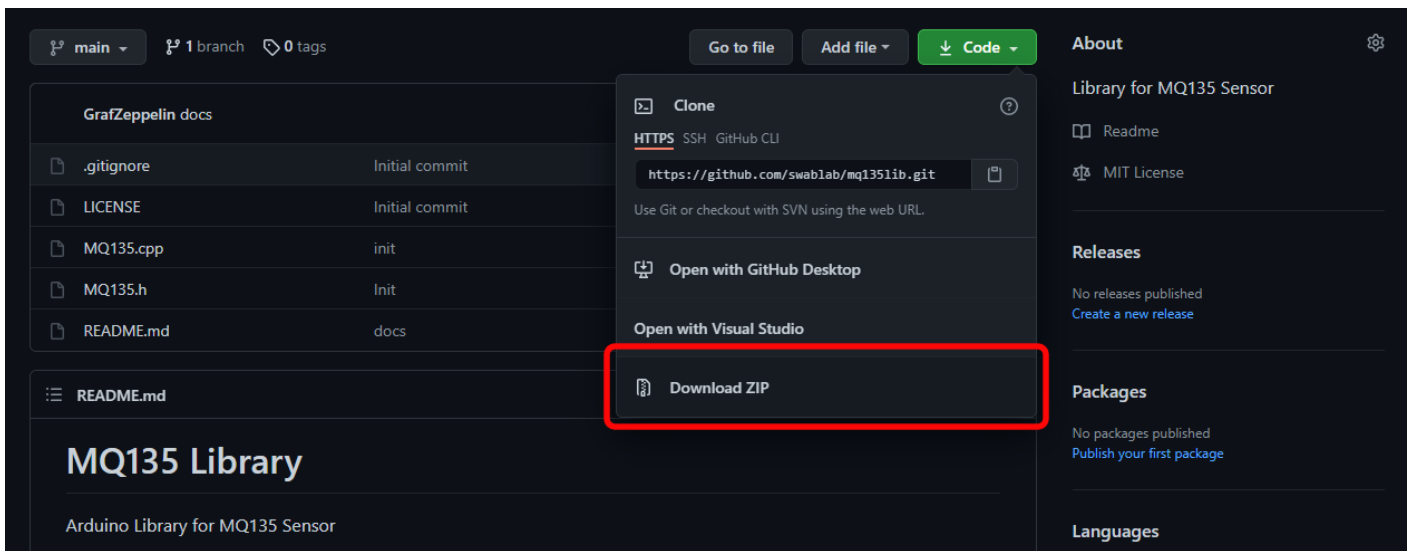
So sind wir also bereits in der Lage, einfache Daten des Sensor auszulesen. Aber wie kommen wir jetzt von diesen Daten auf den CO2-Gehalt der Luft?

Dazu verwenden wir eine Bibliothek für den MQ135, die uns die Werte umrechnet: [MQ135](#)

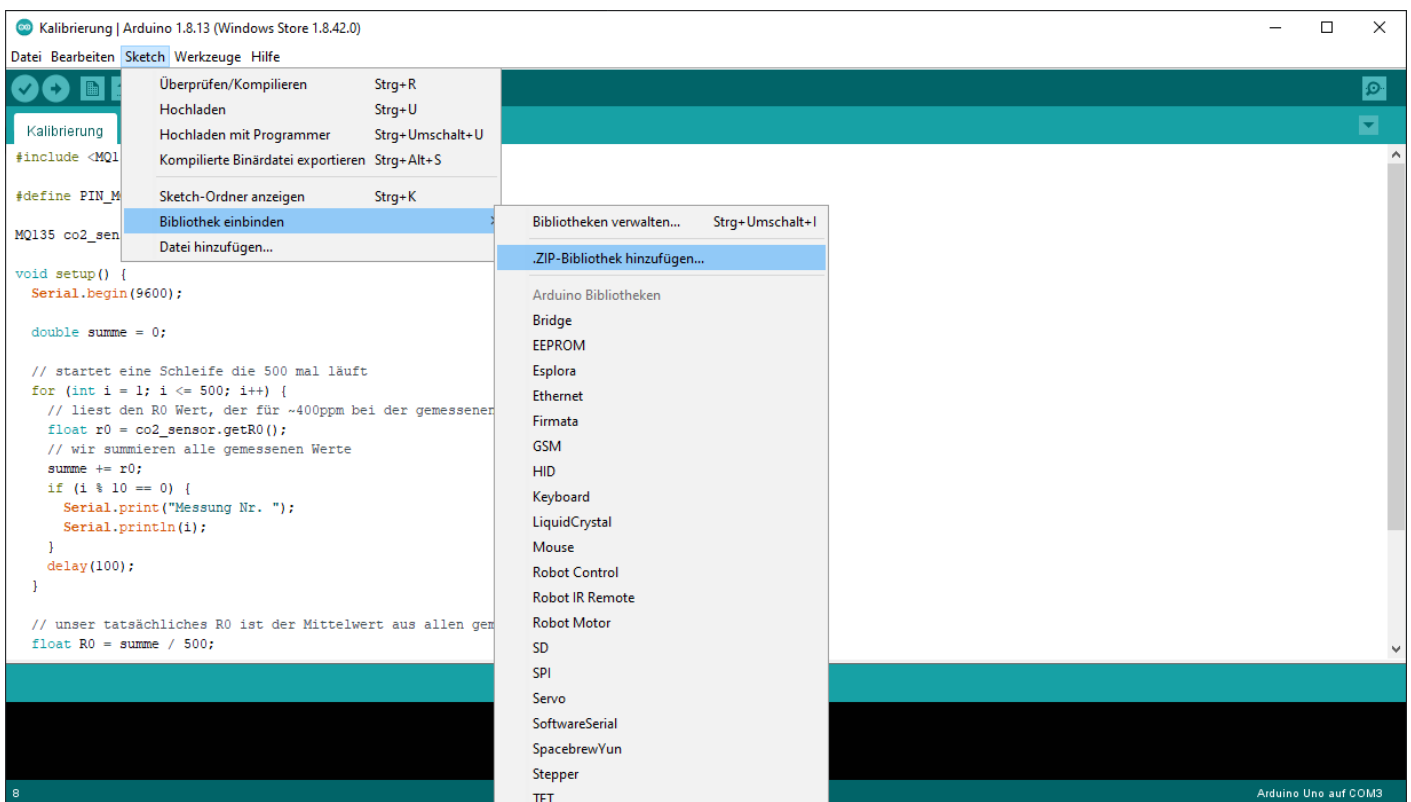
[Bibliothek](#)

Wir binden die Bibliothek in unser Programm ein und können die bereitgestellten Funktionen nutzen.

Die Bibliothek kann als zip heruntergeladen und dann in der IDE importiert werden.



Die Bibliothek als zip herunterladen.



Die Bibliothek kann als zip eingebunden werden.

Damit auf die Funktionen der Bibliothek zugegriffen werden kann, muss sie in das Programm eingebunden werden:

```
#include <MQ135.h>
```

Das ganze Projekt sieht dann so aus:


```
#include <MQ135.h>

#define PIN_MQ135 A0

// Die Bibliothek stellt uns ein MQ135 Objekt zur Verfügung
// Das Objekt wird der Pin übergeben, an dem der Sensor angeschlossen ist.
MQ135 co2_sensor(PIN_MQ135);

void setup() {
    Serial.begin(9600);
}

void loop() {
    // über unser MQ135 Objekt können wir direkt den CO2 Gehalt lesen
    float value = co2_sensor.getCO2();
    Serial.print("co2: ");
    Serial.println(value);
    delay(1000);
}
```

Kalibrierung

Bevor der Sensor richtig zum Einsatz kommt und korrekte Werte liefert, muss er kalibriert werden. Zur Kalibrierung sind folgende Schritte notwendig:

1. Der Sensor muss "einbrennen". Dafür sollte der Sensor zwischen 24 und 48 Stunden einfach angeschlossen sein.
2. Jetzt kann der Sensor kalibriert werden. Wenn kein Vergleichswert von einem anderen CO2-Messer herangezogen werden kann, kann der Sensor an der frischen Luft kalibriert werden. Wir gehen davon aus, dass draußen im Freien der durchschnittliche CO2 Gehalt (von ca. 400ppm) herrscht. Die Library bietet eine Funktion, die den R0 Wert für den gemessenen Widerstand bei ca. 400ppm ausgibt. Wir lassen den Sensor nach dem Einbrennen einige Minuten an der frischen Luft und starten dann die Kalibrierung. Dazu berechnen wir einige R0 Werte und mitteln dann über alle gemessenen Werte. Hierdurch wird unser R0 Wert ermittelt, der exakt für diesen einen, von uns verbauten Sensor gilt. Der Vorgang wird einmalig durchgeführt, danach kann der Sensor verwendet werden.

Das Kalibrierungs-Programm:

```

#include <MQ135.h>

#define PIN_MQ135 A0

MQ135 co2_sensor(PIN_MQ135);

void setup() {
    Serial.begin(9600);

    double summe = 0;

    // startet eine Schleife die 500 mal läuft
    for (int i = 1; i <= 500; i++) {
        // liest den R0 Wert, der für ~400ppm bei der gemessenen Spannung gilt
        float r0 = co2_sensor.getR0();
        // wir summieren alle gemessenen Werte
        summe += r0;
        if (i % 10 == 0) {
            Serial.print("Messung Nr. ");
            Serial.println(i);
        }
        delay(200);
    }

    // unser tatsächliches R0 ist der Mittelwert aus allen gemessenen Werten
    float R0 = summe / 500;

    Serial.print("R0: ");
    Serial.println(R0);
}

void loop() {

}

```

Es werden 500 Werte verwendet, deren Durchschnitt anschließend Anwendung findet. Hieraus resultiert unser Kalibrierungswert R0. Jetzt können wir unser eigentliches Programm schreiben.

```
#include <MQ135.h>

#define PIN_MQ135 A0
#define PIN_LED_GREEN DD2
#define PIN_LED_YELLOW DD3
#define PIN_LED_RED DD4

MQ135 co2_sensor(PIN_MQ135);
float ppm;

void setup() {
    pinMode(PIN_LED_GREEN, OUTPUT);
    pinMode(PIN_LED_YELLOW, OUTPUT);
    pinMode(PIN_LED_RED, OUTPUT);
    pinMode(PIN_NOISE, OUTPUT);
    // Kalibrierung mit dem zuvor berechneten Wert
    co2_sensor.setR0(1000);
}

void loop() {
    ppm = co2_sensor.getCO2();

    if (ppm < 1000) {
        digitalWrite(PIN_LED_GREEN, 1);
        digitalWrite(PIN_LED_YELLOW, 0);
        digitalWrite(PIN_LED_RED, 0);
    }
    else if (ppm <= 2000) {
        digitalWrite(PIN_LED_GREEN, 0);
        digitalWrite(PIN_LED_YELLOW, 1);
        digitalWrite(PIN_LED_RED, 0);
    }
    else {
        digitalWrite(PIN_LED_GREEN, 0);
        digitalWrite(PIN_LED_YELLOW, 0);
        digitalWrite(PIN_LED_RED, 1);
    }

    delay(1000);
}
```

Jetzt fehlt noch der Summer. Dafür müssen wir unser Programm etwas umstellen, da wir neue CO2 Werte nur jede Sekunde einlesen wollen, der Summer jedoch, wenn er aktiv ist, in einem schnelleren Takt als eine Sekunde summen soll.

Wir führen einen Zähler ein, der immer von 0 bis 10 zählt und anschließend misst. Wenn der Wert bei über 2000 ppm liegt aktivieren wir den Summer und schalten ihn in jedem Durchgang abwechselnd ein und aus.

```
#include <MQ135.h>

#define PIN_MQ135 A0
#define PIN_LED_GREEN DD2
#define PIN_LED_YELLOW DD3
#define PIN_LED_RED DD4
#define PIN_NOISE DD5

void printValues();

const int maxCount = 10;
int count = 0;
bool noise = false;
bool noiseActive = false;

float ppm;

MQ135 co2_sensor(PIN_MQ135);

void setup() {
  pinMode(PIN_LED_GREEN, OUTPUT);
  pinMode(PIN_LED_YELLOW, OUTPUT);
  pinMode(PIN_LED_RED, OUTPUT);
  pinMode(PIN_NOISE, OUTPUT);
  Serial.begin(9600);
  co2_sensor.setR0(100);
}

void loop() {
  if (count >= maxCount || count < 0) {
    ppm = co2_sensor.getCO2();
```

```
count = 0;
if (ppm < 1000) {
    noiseActive = false;
    digitalWrite(PIN_LED_GREEN, 1);
    digitalWrite(PIN_LED_YELLOW, 0);
    digitalWrite(PIN_LED_RED, 0);
}
else if (ppm <= 2000) {
    noiseActive = false;
    digitalWrite(PIN_LED_GREEN, 0);
    digitalWrite(PIN_LED_YELLOW, 1);
    digitalWrite(PIN_LED_RED, 0);
}
else {
    noiseActive = true;
    digitalWrite(PIN_LED_GREEN, 0);
    digitalWrite(PIN_LED_YELLOW, 0);
    digitalWrite(PIN_LED_RED, 1);
}

printValues();
}

if (noiseActive) {
    noise = !noise;
    digitalWrite(PIN_NOISE, noise);
}
else {
    digitalWrite(PIN_NOISE, 0);
}

delay(100);
count++;
}

void printValues() {
    Serial.print("ppm: ");
    Serial.println(ppm);
}
```

Wir starten mit dem Zähler bei -1 `int count = -1;`, sodass wir beim ersten Durchgang direkt eine Messung machen. Dafür schreiben wir in die Bedingung der Messung, dass wir entweder messen, wenn der Zähler die Schwelle von 10 überschritten hat, oder wenn der Zähler kleiner als 0 ist:

```
if (count >= maxCount || count < 0)
```

Wenn der CO2 Gehalt größer als 2000 ppm ist, aktivieren wir den Summer (`noiseActive = true;`) und schalten diesen bei jedem Durchgang ein oder aus:

```
noise = !noise;  
digitalWrite(PIN_NOISE, noise);
```

Zum Schluss dürfen wir nicht vergessen zu warten und den Zähler zu erhöhen:

```
delay(100);  
count++;
```

Nach einer Messung müssen wir den Zähler wieder auf 0 zurück setzen `count = 0;`

Der Quellcode ist auch in unserem Github Repository zu finden: [Quellcode](#).

Die Bibliothek für den MQ135 befindet sich ebenfalls auf Github: [Bibliothek](#)

Optisches Tuning

Unser CO2 Melder hat inzwischen alle Funktionalitäten, um den CO2-Gehalt der Luft zu Messen und beim Überschreiten eines kritischen Schwellwertes dies über optische und/oder akustische Signale zu melden. Einen Schönheitspreis würde er aber noch nicht gewinnen.

Platine

Der ganze Prototypen-Aufbau mit einem Arduino Mega, einem Breadboard und Steckkabelverbindung ist platzintensiv und fehleranfällig. Es reicht bereits ein einfacher Wackelkontakt eines Kabels, dass LEDs nicht mehr leuchten oder die Werte eines Sensor nicht mehr korrekt ermittelt werden können.

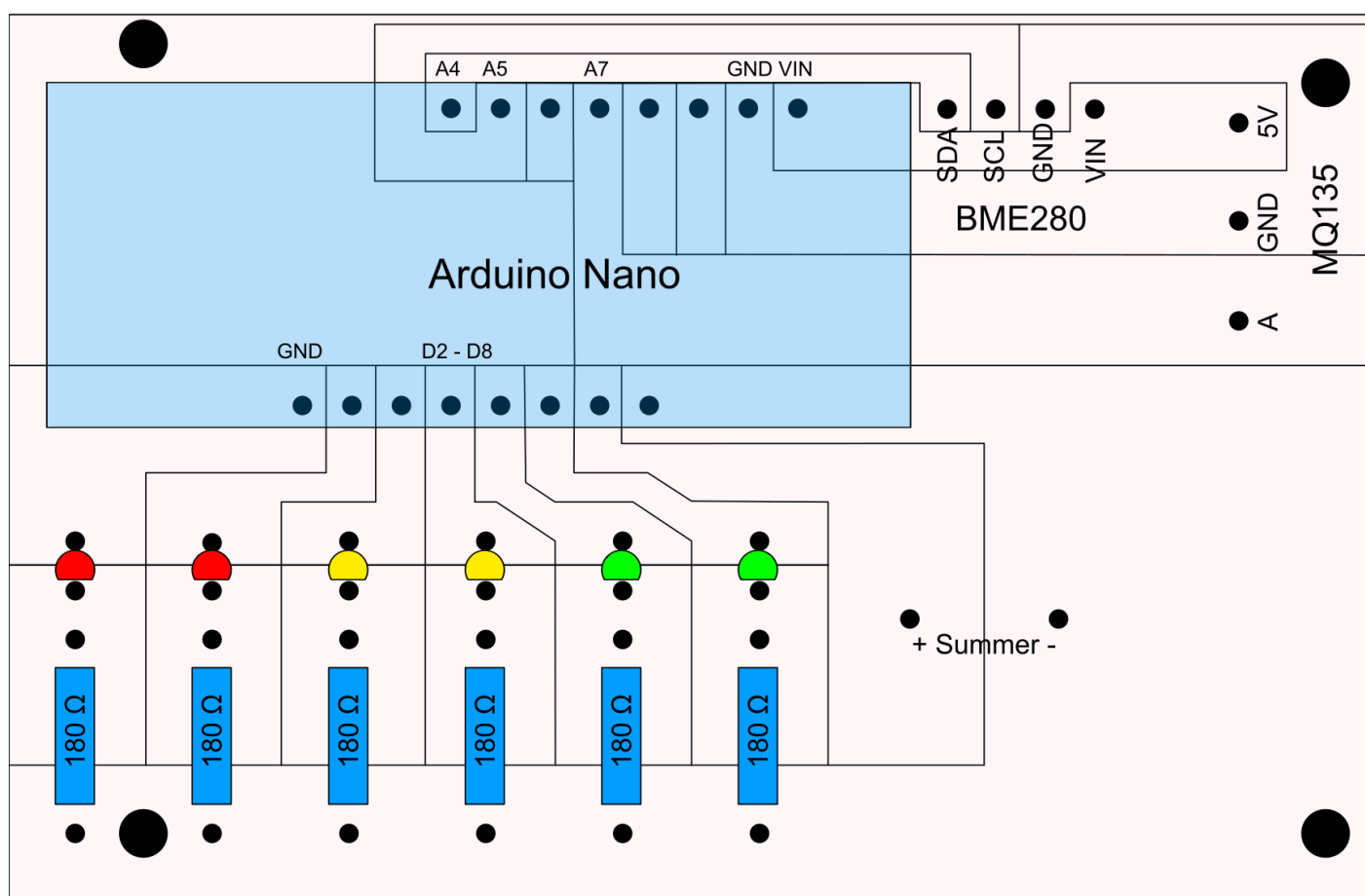
Diese beiden Nachteile können durch verwenden eines Arduino Nano (sehr kleiner Mikrocontroller) und einer festen Verlotung der Bauteile auf einer Platine beseitigt werden.

Um Platinen herzustellen gibt es unterschiedliche Vorgehensweisen. Als Grundlage dient oft eine Kunststoffplatte, die auf einer Seite mit einer sehr dünnen Kupferschicht überzogen ist. Um die elektronischen Bauteile auf der Platine zu befestigen, werden an den entsprechenden Stellen Löcher in die Platine gebohrt. Um Leiterbahnen zur Verbindung der Löcher zu erstellen, wird die Kupferschicht an den Rändern der Leiterbahnen entfernt.

Zwei gängige Verfahren zum Auftrennen der Kupferschicht in einzelne Leiterbahnen sind beispielsweise fräsen und ätzen.

Wir haben uns dazu entschieden die Leiterbahnen auf unserer Platine durch fräsen zu trennen. Dazu entwirft man zunächst eine Vektorgrafik, welche die Bohrungen für die Bauteile beinhaltet. Anschließend fügt man Linien als Abtrennung der Leiterbahnen ein.

Wir haben Affinity Designer zur Entwicklung dieser Vektorgrafik genutzt, es gibt aber auch freie Alternativen, mit denen ebenfalls Vektorgrafiken erstellt werden können, beispielsweise Inkscape. Nachfolgende Grafik zeigt unser entworfenes Schema für die Platine:

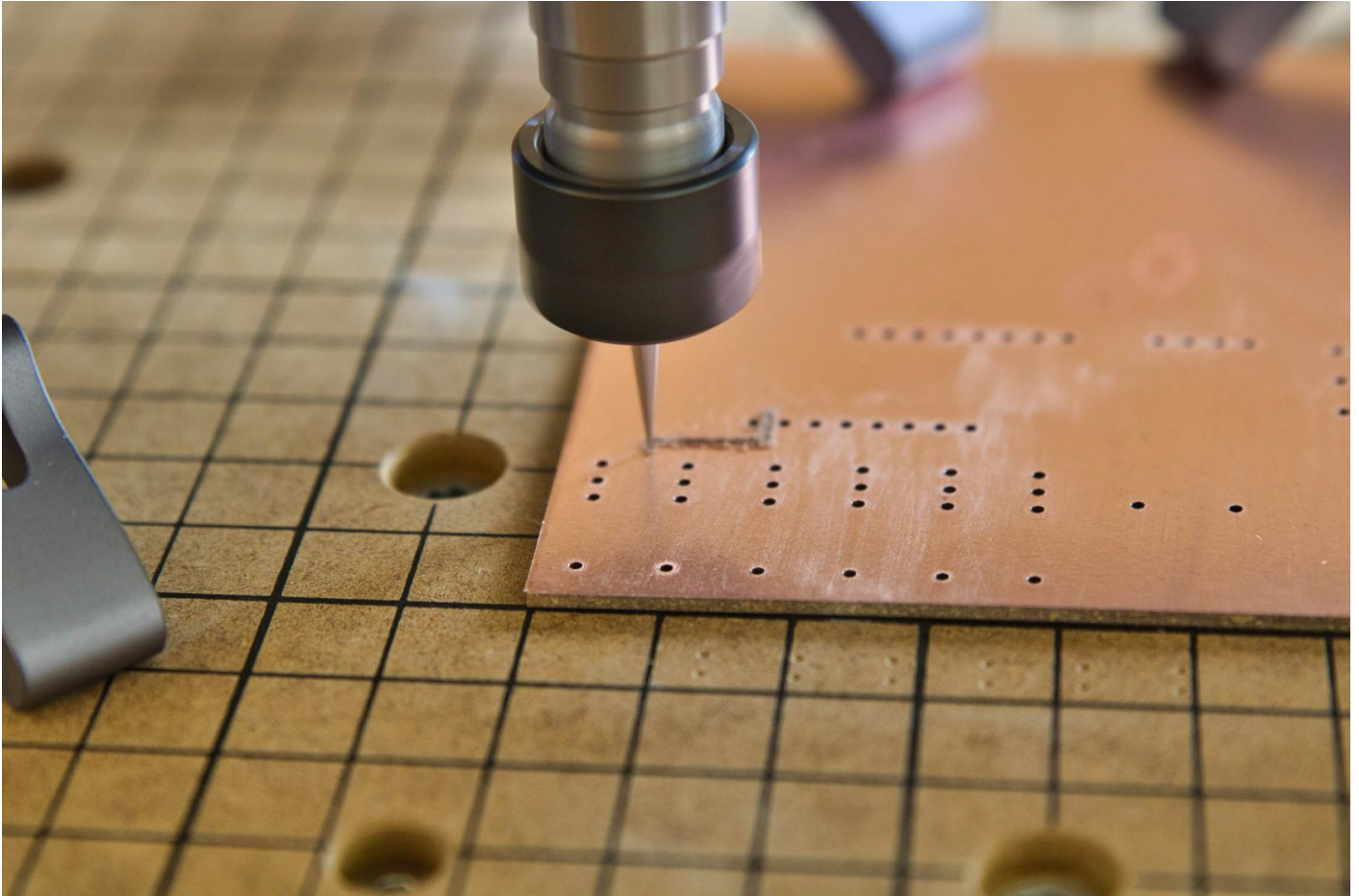


Die folgenden SVG-Dateien enthalten nur die Inhalte aus diesem Platinenschema, die für die jeweilige Bearbeitung relevant ist:

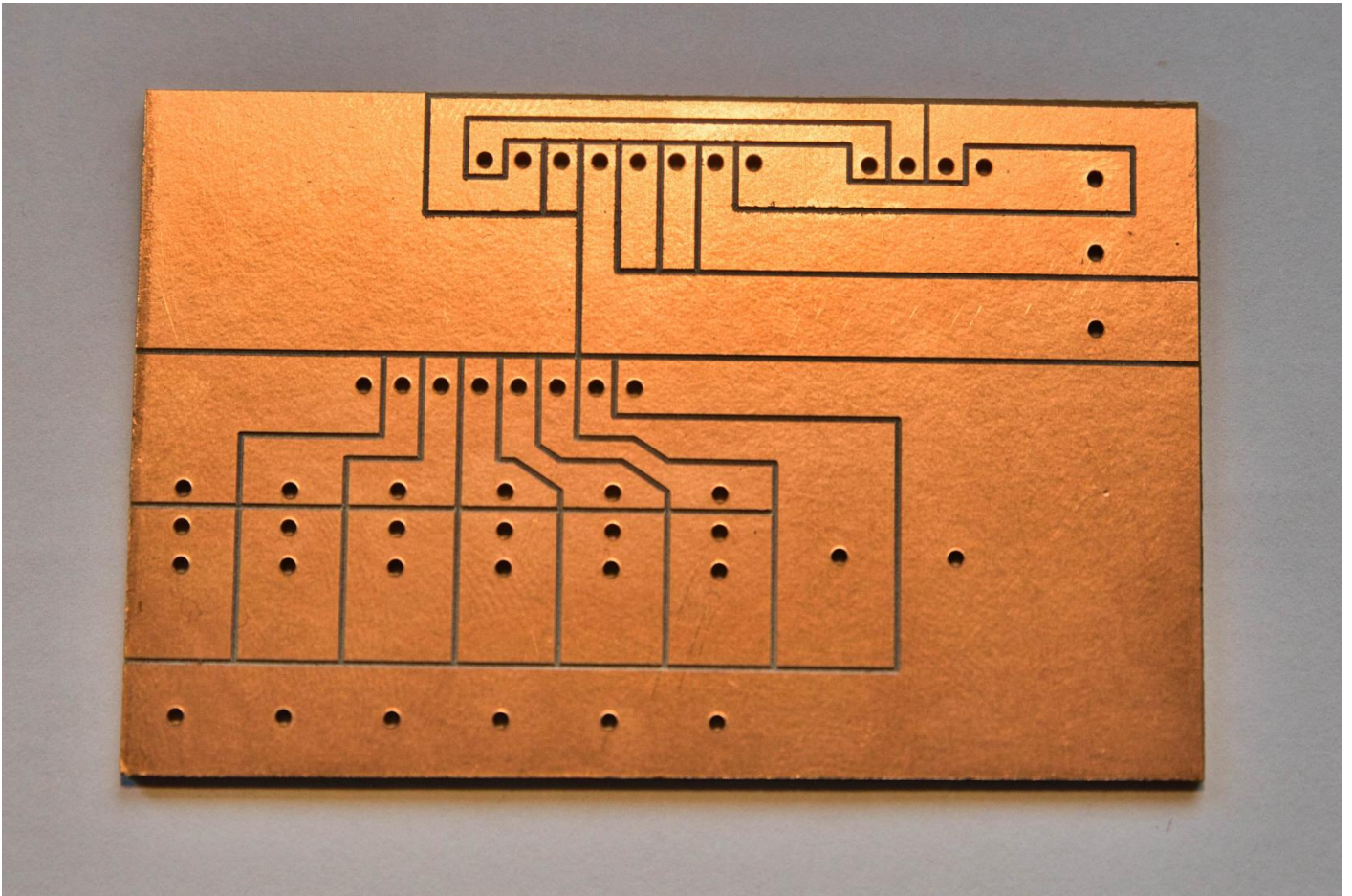
Fräsbearbeitung	Download-Link	Fräswerkezeug
Leiterbahnen	platine_co2_4_leiterbahnen.svg	Gravurstichel
Bohrungen für Lötstellen	platine_co2_4_loetstellen.svg	Bohrer 1 mm

Bohrungen für die Befestigung des Gehäuses	platine_co2_4_bohrung_gehaeuse.svg	Bohrer 2,5 mm
--	--	---------------

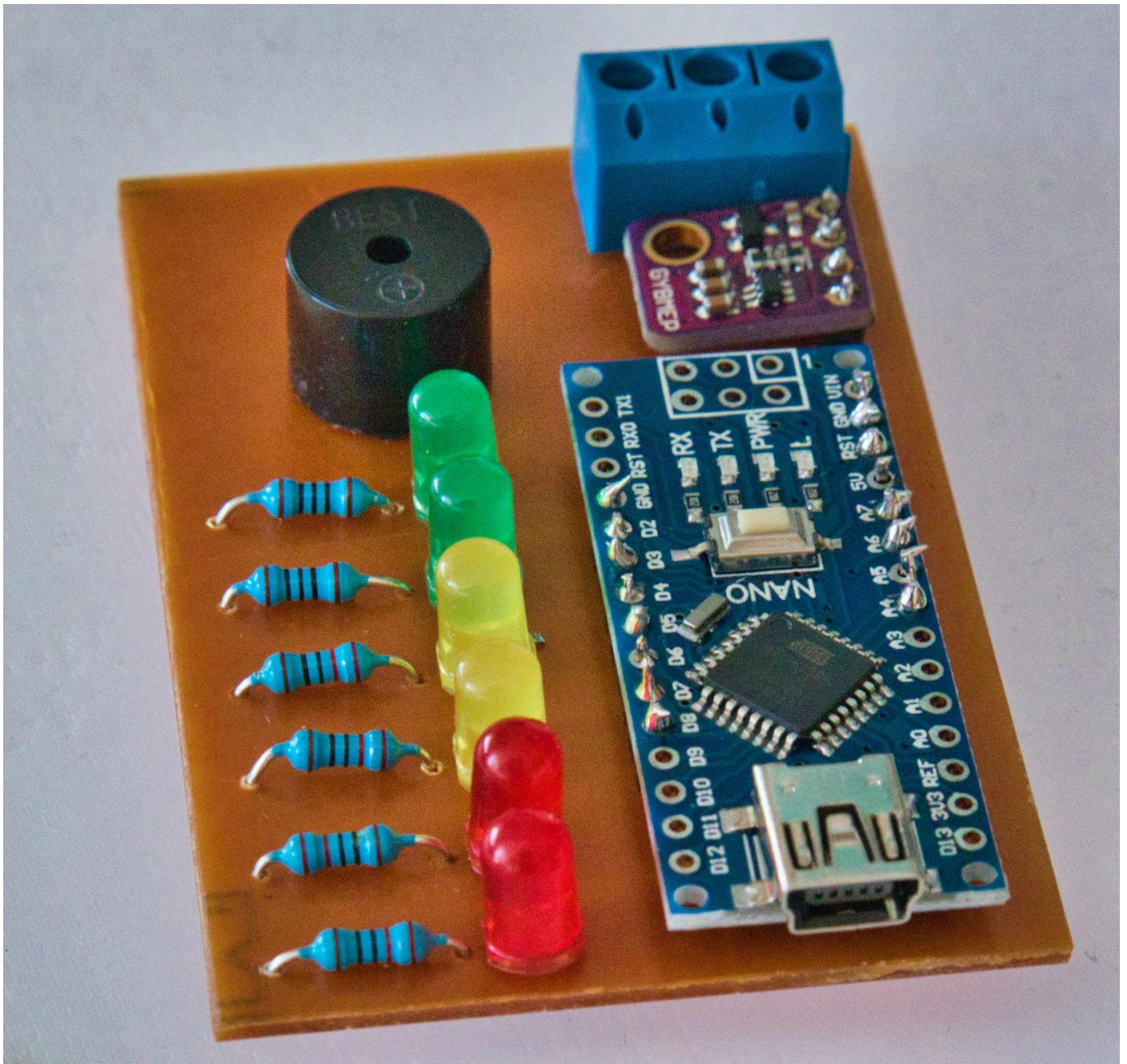
Aus diesen Vektorgrafiken kann nun ein NC-Code generiert werden. Mit dem erzeugten NC-Code weiß die CNC-Fräsmaschine, wo welche Bearbeitungen vorgenommen werden sollen. Da wir einen Snapmaker 2.0 zum fräsen verwenden, nutzen wir zum erzeugen der Fräsbahnen und Bohrungen die Software [Snapmaker Luban](#).



Nach beendetem Fräsvorgang sieht die Platine so aus.



Als nächstes muss die Platine mit den Bauteilen bestückt und die Anschlüsse an die Platine gelötet werden. Hierzu kann man obiges Platinenschema als Hilfe verwenden.
Nach erfolgreicher Verlötung sieht die endgültige Platine dann so aus:



Gehäuse

Um die selbst gebaute Platine vor Staub und sonstigen Umwelteinflüssen zu schützen, wird ein Gehäuse entworfen. Dieses soll möglichst klein sein, um den CO2 Melder jederzeit an anderen Standorten einsetzen zu können. Da wir die Möglichkeit haben, auf 3D Drucker zurückzugreifen, liegt es nahe, auch dieses Gehäuse mit dem 3D-Druckverfahren zu fertigen. Hierfür werden sämtliche Bauteile im CAD erstellt. Anschließend wird das Gehäuse gedruckt. Das Ergebnis sieht wie folgt aus:



Falls kein 3D Drucker verwendet werden kann, kann selbstverständlich jedes andere passende Gehäuse verwendet werden.

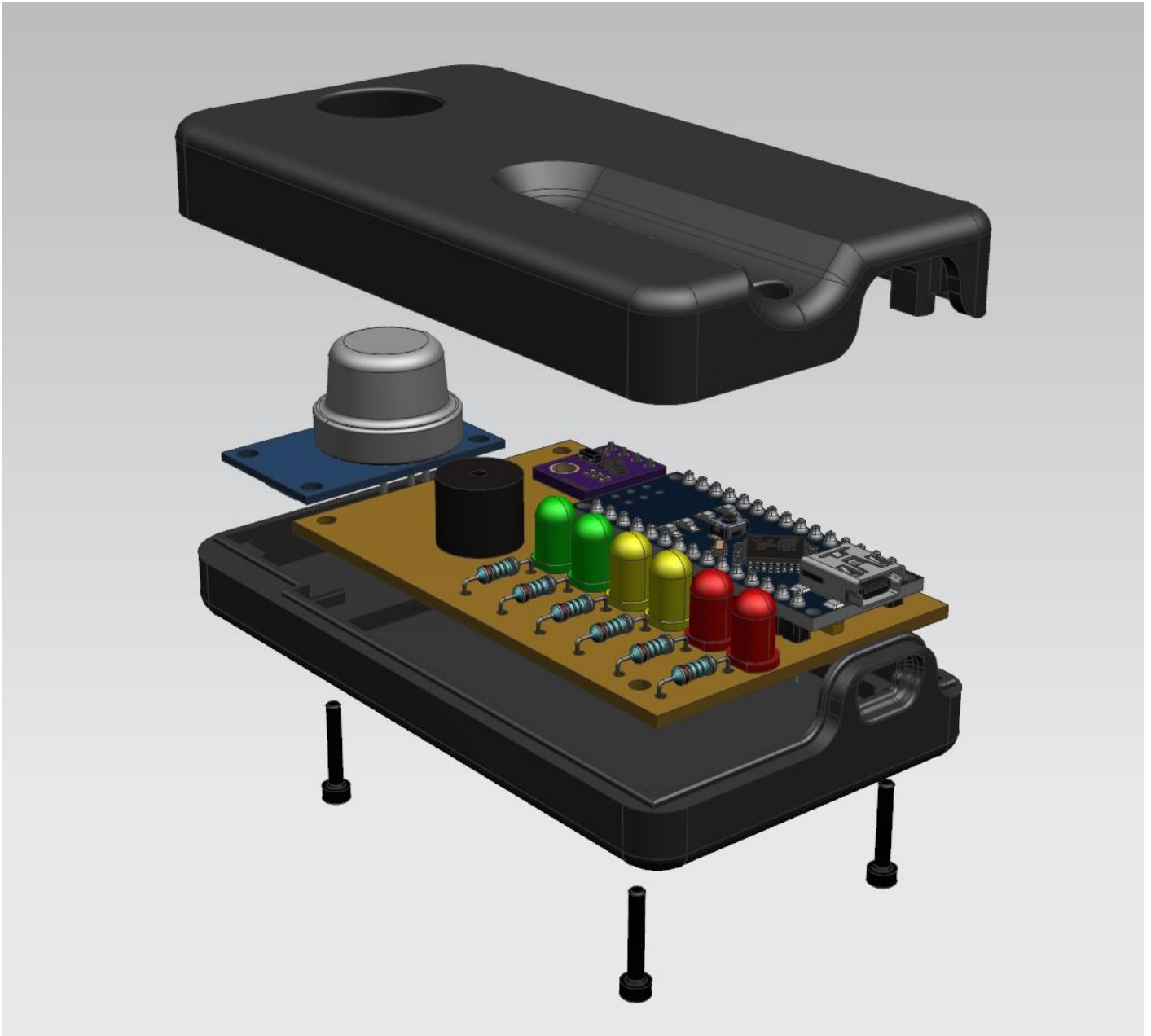
Die Dateien für das Gehäuse können hier heruntergeladen werden:

<https://www.prusaprinters.org/de/prints/64081-co2-detector> (Step vorhanden, um nötige Änderungen für euren Anwendungsfall möglich zu machen).

Zusammenbau

Der Zusammenbau gestaltet sich sehr einfach. Bei Bedarf sollten zuerst die 3D Druck Teile gesäubert werden. Anschließend kann die Platine in das Gehäuse eingelegt werden. Hierbei muss darauf geachtet werden, dass zuerst der USB-Anschluss des Arduinos in dessen vorgesehene Position geschoben wird und anschließend die gesamte Platine im Gehäuse versenkt wird.

Anschließend kann der MQ135 Sensor in das Gehäuse eingelegt werden. Nun kann der Gehäusedeckel auf das Unterteil des Gehäuses gelegt werden und mit vier Zylinderschrauben fixiert werden.



Jetzt ist der CO2 Melder für seinen vorgesehen Einsatzzweck bereit.

Bei den mit "*" markierten Links handelt es sich um Affiliate-Links.

Wenn ihr etwas über diese Links kauft, erhalten wir eine kleine Provision. Für euch ändert sich dabei nichts.

Dadurch könnt ihr uns ohne Aufwand unterstützen!

Wo ihr eure Sachen kauft, bleibt aber natürlich euch überlassen!

Version #3

Erstellt: 25 Februar 2025 20:27:20 von Manuel

Zuletzt aktualisiert: 25 Februar 2025 20:38:05 von Manuel