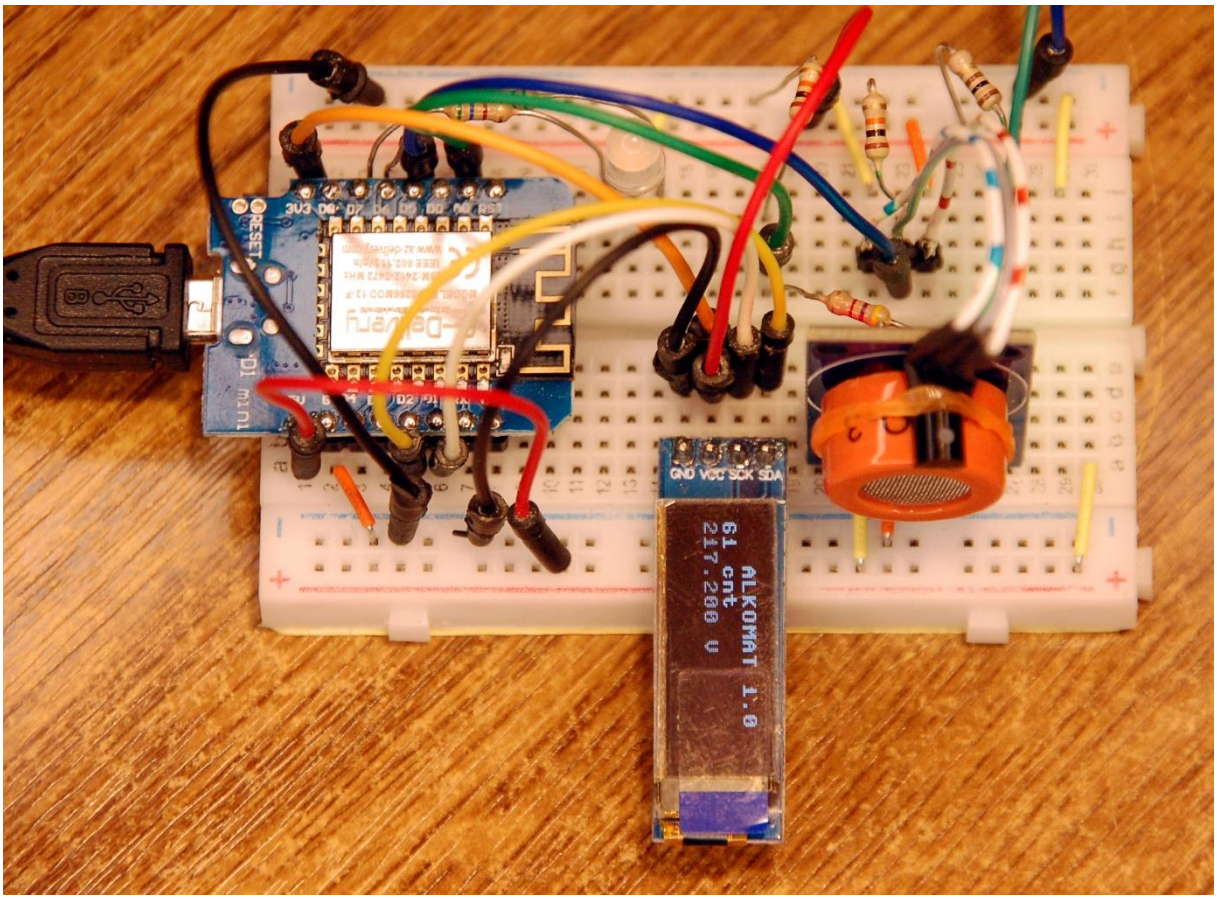

Alkomat - Aufbau

Diese Blogfolge gibt es auch als PDF-Dokument.

Die Faschingszeit naht und da wird – die Pandemie ist vorüber, sagt man – sicher hier und da eine Fete angesagt sein. Und wo gefeiert wird, wird auch getrunken, sicher nicht nur Brause und Saft, sondern auch ganz andere Sachen. Aber bevor man sich hinters Steuer setzt, sollte man vorsichtshalber den Blutalkoholspiegel im Visier behalten, andernfalls hat der Spaß schnell ein Loch, wenn der Bußgeldbescheid ins Haus flattert, wegen Trunkenheit im Straßenverkehr. Wie Sie messtechnisch Ihren Alkoholkonsum überprüfen können, das zeigt der zweite Beitrag aus der Reihe

MicroPython auf dem ESP32 und ESP8266

heute

Ein Alkometer mit dem ESP8266 und einem MQ-3-Gas-Sensor – Teil 2

In diesem Teil geht es neben dem Programm auch um die Hintergründe für die Auswertung eines Diagramms, wie man es im Datenblatt des MQ-3 findet. Schließlich brauchen wir eine Formel, die es erlaubt, die gemessene

Sensorspannung in eine Alkoholkonzentration in Luft umzusetzen. Dazu ist ein wenig Mathematik notwendig.

Die technischen und programmbedingten Hintergründe haben wir ja bereits im Teil 1 zum Thema Alkometer behandelt. An der Schaltung hat sich nichts geändert, auch nicht an den verwendeten Bausteinen. Die Funktionen aus Teil 1 werden in dieser Folge ebenfalls recycled. Ein paar neue sind dazugekommen, die bespreche ich weiter unten. Für die Eichung des Geräts bleibt aus den schon genannten Gründen letztlich nur der Selbstversuch. Hier noch einmal die Vorgehensweise.

Nach einem Blick in den Bußgeldkatalog könnte das so aussehen. Verdoppelt man den Wert der Atemluftkonzentration, erhält man den Blutalkoholspiegel. Andersherum kriege ich die Atemluftkonzentration durch Halbieren des Blutwerts. Letzteren kann man berechnen. Ein Beispiel, dieses Mal weiblicher Natur:

Eine Frau mit 70kg Körpermasse besitzt 70kg • 0,6 = 42kg Körperflüssigkeit. Der Reduzierfaktor bei Männern ist 0,7. Sie trinkt 0,5L Wein = zwei Schoppen mit einem Alkoholgehalt von 12,5 % Vol. Im Wein sind also 500cm³ • 12,5 / 100 = 62,5cm³ reiner Alkohol. Reiner Alkohol hat eine Dichte von 0,79g/cm³, damit haben die 62,5cm³ eine Masse von 42,5cm³ • 0,79g/cm³ = 49,4g. Bezogen auf 42kg Körperflüssigkeit sind das 49,4g / 42kg = 1,18 Promille. Und das würde einer Atemluftkonzentration von knapp 0,59 Promille oder 0,59mg Alkohol / Liter Atemluft ergeben. Allerdings müsste der Wein möglichst in einem Zug getrunken werden, denn die Leber baut pro Stunde 0,15 Promille ab. Ob Sie jedoch nach dem Genuss von zwei Schoppen Wein noch in der Lage sind, Ihren Aufbau zu calibrieren, können nur Sie selbst entscheiden, Autofahren sollten Sie dann besser nicht mehr.

Hardware

Als Controller habe ich einen ESP8266 gewählt, weil der eine einigermaßen lineare ADC-Kennlinie hat. Grundsätzlich wäre auch ein ESP32 brauchbar, aber der verursacht einen höheren Aufwand für das Geraderichten der Kennlinie. Ich komme später darauf zurück.

Die ersten drei ESP8266-Modelle in der Teileliste haben den Vorteil, dass am Anschluss VIN / 5V die Spannung des USB-Anschlusses verfügbar ist. Das spart während der Entwicklung ein zusätzliches Netzteil oder eine Batterie. Für den späteren Betrieb als Stand-Alone-Gerät ist ein Netzteil empfehlenswert, denn Batterien oder ein Akku ist schnell leergelutscht, immerhin zieht die Heizung des MQ-3 150 mA und der ESP8266 noch einmal 20 mA.

1	D1 Mini NodeMcu mit ESP8266-12F WLAN Modul oder D1 Mini V3 NodeMCU mit ESP8266-12F oder NodeMCU Lua Amica Modul V2 ESP8266 ESP-12F WIFI oder NodeMCU Lua Lolin V3 Module ESP8266 ESP-12F WIFI
1	0,91 Zoll OLED I2C Display 128 x 32 Pixel
1	Alkohol Gas Sensor DC 5V MQ-3 mit Signalausgang
1	Mehrgang rotary Potentiometer mit Schutzwiderstand 3590S 10K Ohm
1	KY-011 Bi-Color LED Modul 5mm
1	DS18B20 digitaler Temperatursensor TO92

1	Mini Breadboard 400 Pin mit 4 Stromschienen
	Diverse Jumperkabel
1	Widerstand 4k7
2	Widerstand 10k
2	Widerstand 390 Ohm
1	Battery Expansion Shield 18650 V3 inkl. USB Kabel

Die Software

Fürs Flashen und die Programmierung des ESP32:
Thonny oder
µPyCraft

Verwendete Firmware für den ESP8266:

v1.19.1 (2022-06-18) .bin

Verwendete Firmware für den ESP32:

v1.19.1 (2022-06-18) .bin

Die MicroPython-Programme zum Projekt:

ssd1306.py Hardwaretreiber für das OLED-Display
oled.py API für das OLED-Display
calibrate.py Programm zum Calibrieren des ADC
aufheizen.py Programm zum Ermitteln der Vorheizdauer des MQ-3
alktest.py Programm zum Testen des Atemalkoholpegels

MicroPython - Sprache - Module und Programme

Zur Installation von Thonny finden Sie hier eine ausführliche Anleitung (english version). Darin gibt es auch eine Beschreibung, wie die Micropython-Firmware (Stand 18.06.2022) auf den ESP-Chip gebrannt wird.

MicroPython ist eine Interpretersprache. Der Hauptunterschied zur Arduino-IDE, wo Sie stets und ausschließlich ganze Programme flashen, ist der, dass Sie die MicroPython-Firmware nur einmal zu Beginn auf den ESP32 flashen müssen, damit der Controller MicroPython-Anweisungen versteht. Sie können dazu Thonny, µPyCraft oder esptool.py benutzen. Für Thonny habe ich den Vorgang hier beschrieben.

Sobald die Firmware geflasht ist, können Sie sich zwanglos mit Ihrem Controller im Zwiegespräch unterhalten, einzelne Befehle testen und sofort die Antwort sehen, ohne vorher ein ganzes Programm kompilieren und übertragen zu müssen. Genau das stört mich nämlich an der Arduino-IDE. Man spart einfach enorm Zeit, wenn man einfache Tests der Syntax und der Hardware bis hin zum Ausprobieren und

Verfeinern von Funktionen und ganzen Programmteilen über die Kommandozeile vorab prüfen kann, bevor man ein Programm daraus strickt. Zu diesem Zweck erstelle ich auch gerne immer wieder kleine Testprogramme. Als eine Art Makro fassen sie wiederkehrende Befehle zusammen. Aus solchen Programmfragmenten entwickeln sich dann mitunter ganze Anwendungen.

Autostart

Soll das Programm autonom mit dem Einschalten des Controllers starten, kopieren Sie den Programmtext in eine neu angelegte Blankodatei. Speichern Sie diese Datei unter boot.py im Workspace ab und laden Sie sie zum ESP-Chip hoch. Beim nächsten Reset oder Einschalten startet das Programm automatisch.

Programme testen

Manuell werden Programme aus dem aktuellen Editorfenster in der Thonny-IDE über die Taste F5 gestartet. Das geht schneller als der Mausklick auf den Startbutton, oder über das Menü **Run**. Lediglich die im Programm verwendeten Module müssen sich im Flash des ESP32 befinden.

Zwischendurch doch mal wieder Arduino-IDE?

Sollten Sie den Controller später wieder zusammen mit der Arduino-IDE verwenden wollen, flashen Sie das Programm einfach in gewohnter Weise. Allerdings hat der ESP32/ESP8266 dann vergessen, dass er jemals MicroPython gesprochen hat. Umgekehrt kann jeder Espressif-Chip, der ein kompiliertes Programm aus der Arduino-IDE oder die AT-Firmware oder LUA oder … enthält, problemlos mit der MicroPython-Firmware versehen werden. Der Vorgang ist immer so, wie hier beschrieben.

Der ALK-Tester

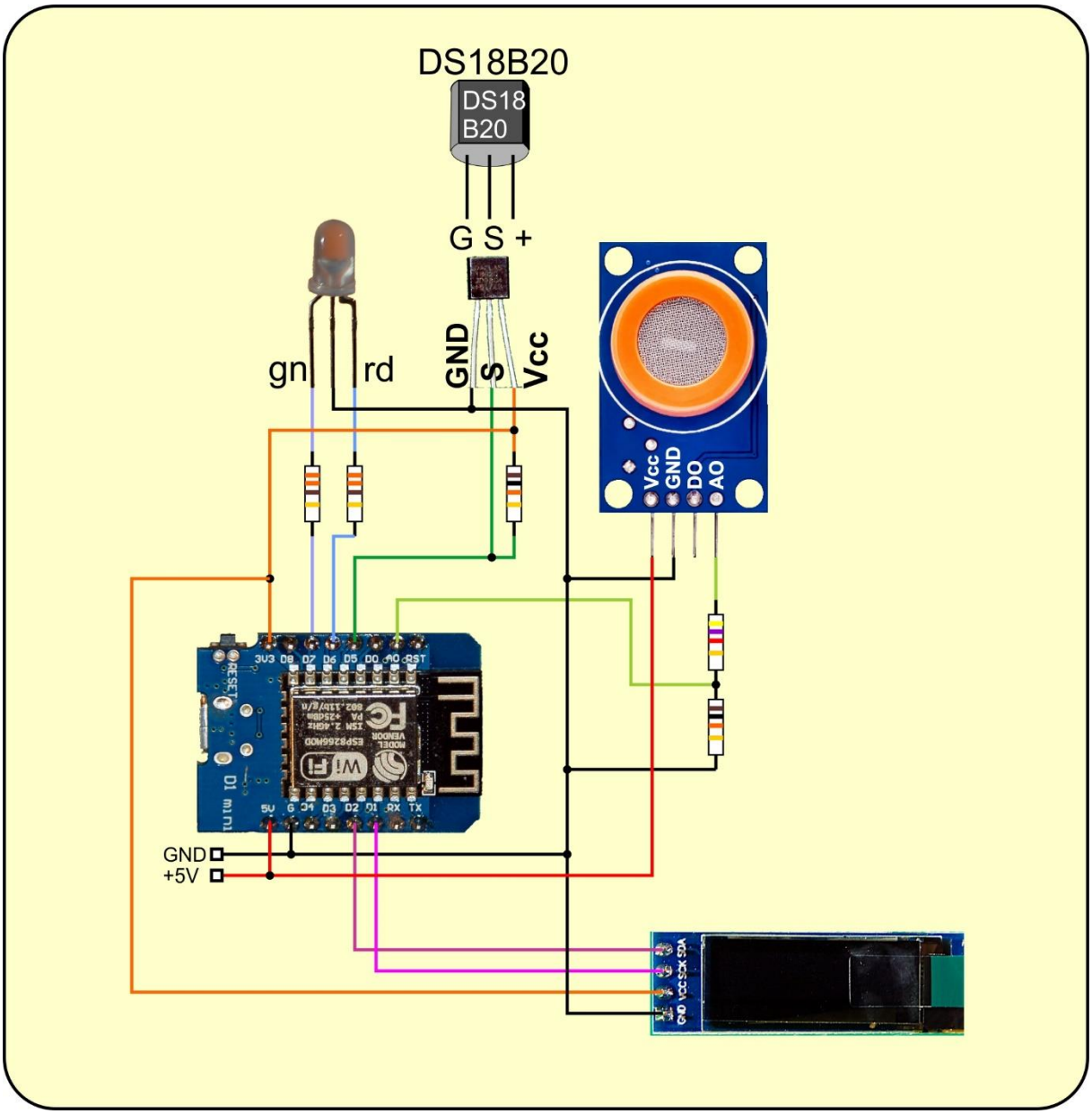

Abbildung 1: Alkomat - Schaltung

Hier noch einmal die Schaltung. Beschrieben habe ich sie bereits ausführlich in Teil 1. Dort finden Sie auch eine Beschreibung der Objekte, Funktionen und Variablen, welche das im Folgenden beschriebene Hauptprogramm nutzt.

Es gibt 4 neue Funktionen, die den bisherigen Block ergänzen. **getRatio**() nimmt den Sensorwiderstand **Rs** und gibt das Verhältnis **Rs/R0** zurück.

```
def getRatio(rs):
    return rs/R0
```

```
def getConcentration(ratio):
    return (0.51959/ratio)**(1/0.679907)
```

Die Funktion **getConcentration**() berechnet aus dem Verhältnis der Widerstände Rs und Ro die Atem-Alkoholkonzentration in Promille. Als Grundlage für die Formel dient das Diagramm aus dem Datenblatt des MQ-3. Ro ist der Sensorwiderstand wenn dieser einer Konzentration von 0,4mg reinem Alkohol in 1L reiner Luft ausgesetzt wird, so sagt es das Datenblatt.

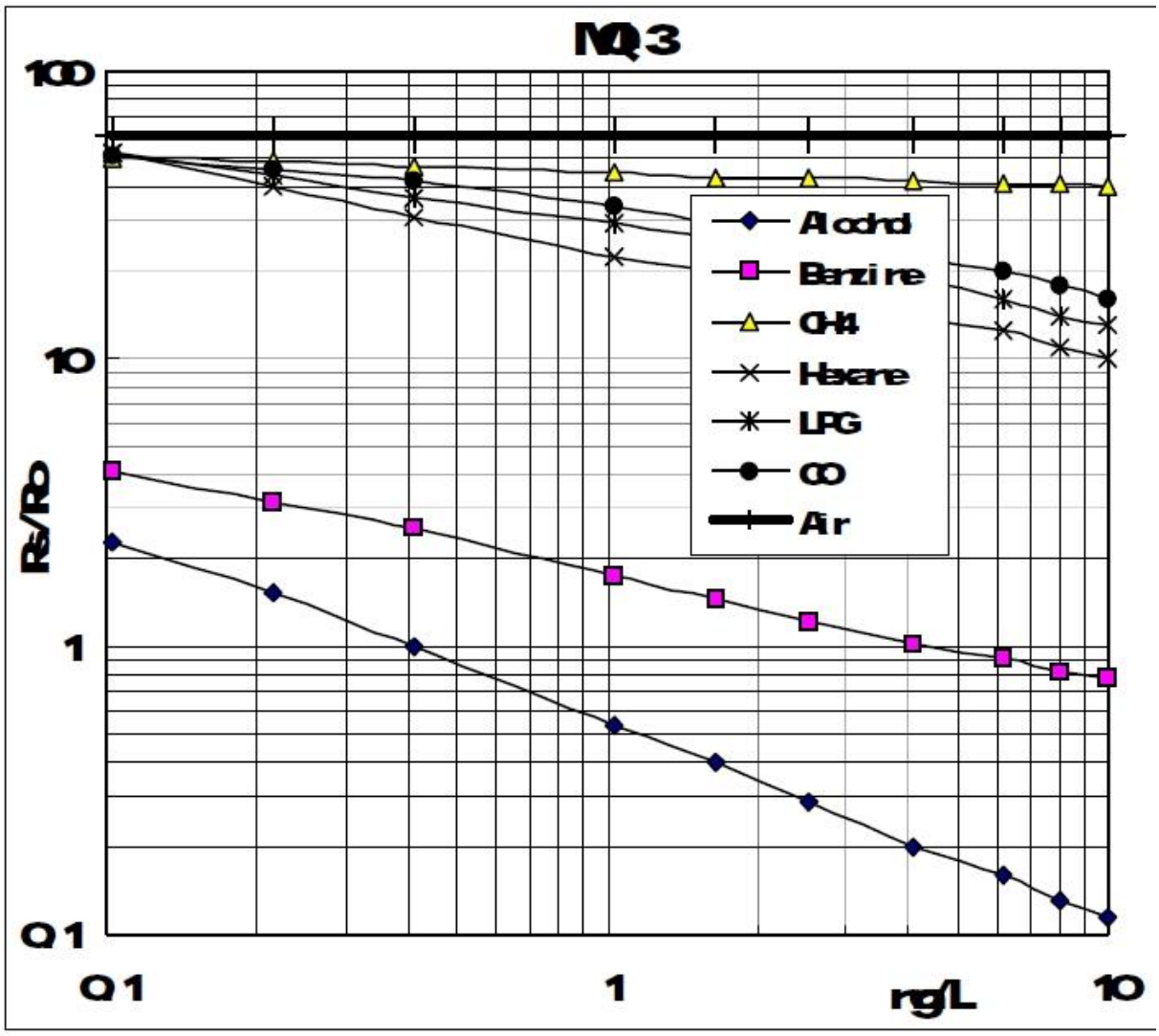

Fig.2 sensitivity characteristics of the MQ-3

Abbildung 2: Vom Widerstandsverhältnis zur Konzentration

Die unterste Kurve ist die von Alkohol. Beide Achsen des Koordinatensystems sind logarithmisch geteilt, und die Punkte liegen annähernd auf einer Geraden. Daraus kann man schließen, dass sich der Zusammenhang durch eine Formel mit der allgemeinen Darstellung $y = A \cdot x^B$ darstellen lässt. Ich habe die Werte aus dem Diagramm abgelesen und in eine Tabelle in Libre Office Calc eingegeben.

conc	ratio
0,1	2,4
0,22	1,45
0,4	1
1	0,52
1,6	0,4
2,4	0,29
4	0,2
6	0,14
8	0,13
10	0,11

Daraus wird das Diagramm mit linear skalierten Achsen.

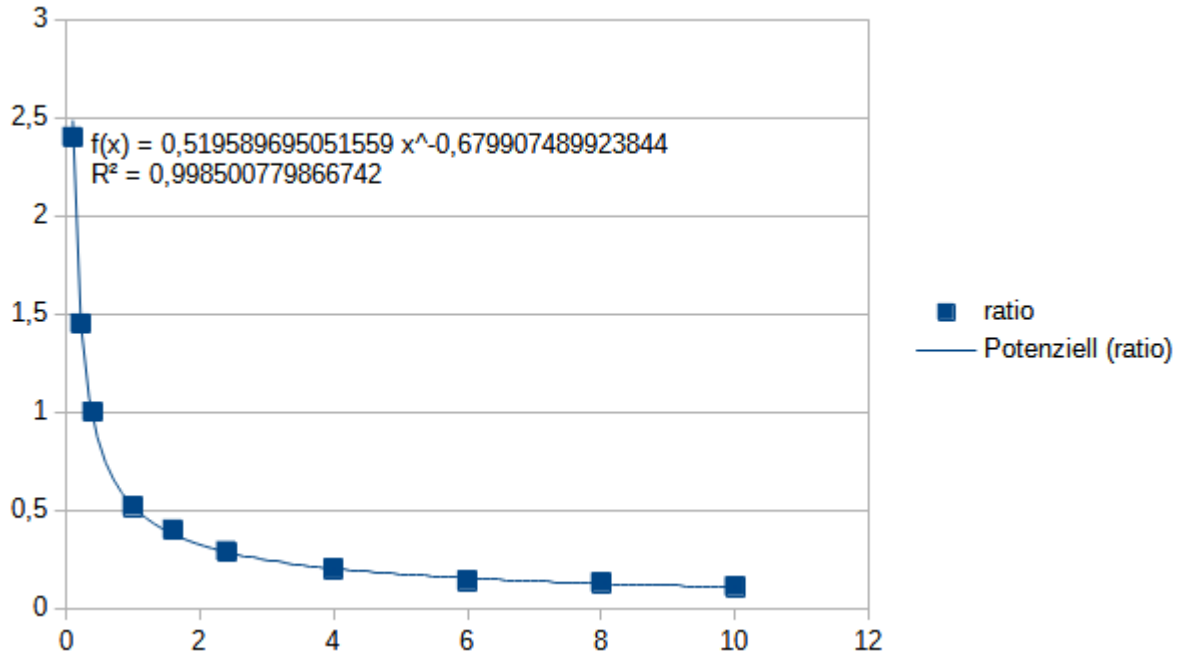

Abbildung 3: Widerstandsverhältnis versus Konzentration

Wie kommt nun Calc auf die Koeffizienten A und B des Funktionsterms? Ich stelle die Achsen auf logarithmische Darstellung um, und schon wird aus dem Graphen wieder eine Gerade. Die Dekaden der logarithmischen Skalierung entsprechen den linearen Einheiten in dem grünen Koordinatensystem der Gerade.

Der Logarithmus von 1 zu einer beliebigen Basis ist stets 0. Dann ist auch B * 0 = 0 und übrig bleibt als Achsenabschnitt t der Geraden der log(A) = 0,52. Den Steigungsfaktor B bekomme ich aus dem Quotienten $\Delta Y / \Delta X$. Als ΔX wähle ich eine Einheit auf der X-Achse, also 1. Die linke Kathete des Steigungsdreiecks muss auch als Vielfaches der y-Einheit Ey ausgedrückt werden. Ich berechne ΔY also einfach als Quotient aus den Streckenlängen der Kathete und der Einheit Ey in Millimetern zu 0,678.

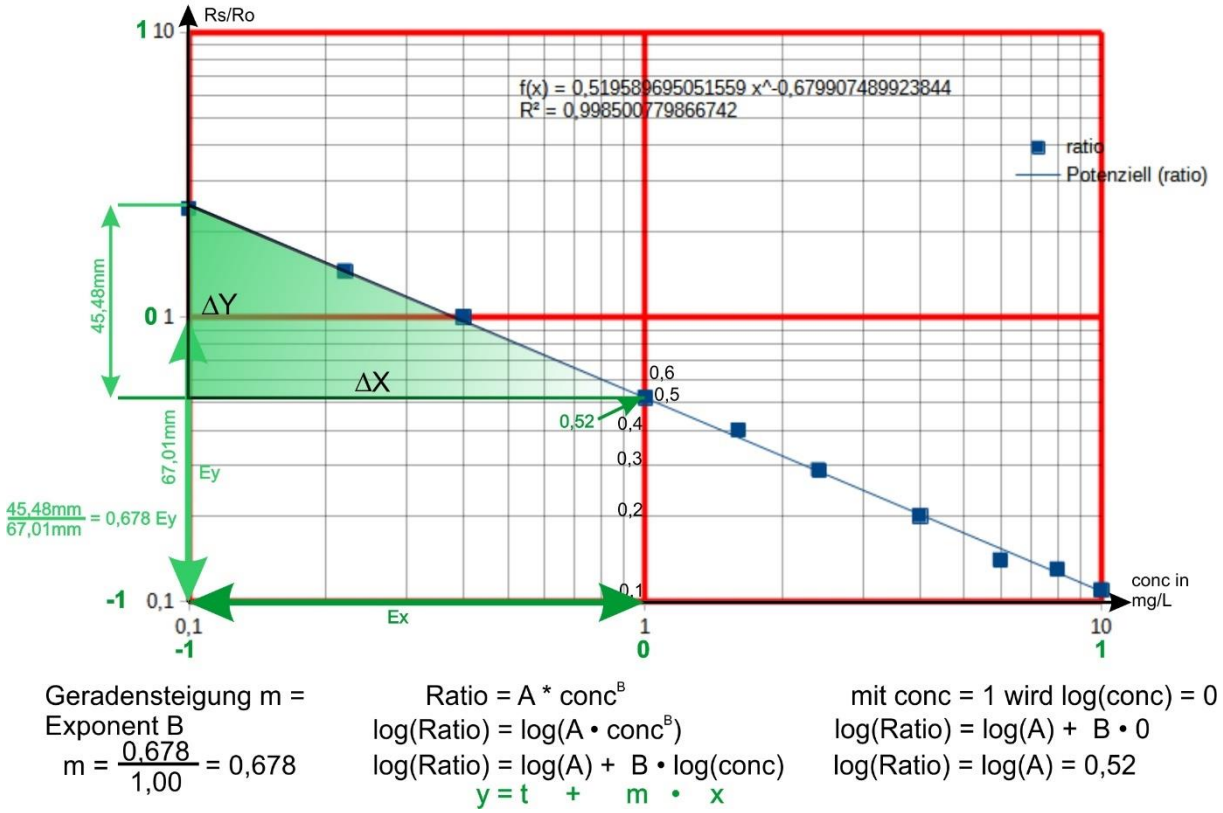

Abbildung 4: Diagrammauswertung

Der Exponent ist negativ, weil die Gerade fällt. Die Gleichung Ratio = A • conc $^{-B}$ löse ich nach conc auf und erhalte conc = (A / Ratio) $^{(1 / B)}$. Diese Gleichung benutzt die Funktion **getConcentration**(). Ich bin somit in der Lage, aus dem einen Messwert am Sensorausgang zunächst den Sensorwiderstand Rs zu berechnen, mit diesem Wert und dem Lastwiderstand RL erhalte ich das Verhältnis Rs/Ro und hiermit schließlich die Atemluftkonzentration.

```
def awaitRunTemp(chip):
    temp=getTemperature(chip)
    while temp < tmin:
        temp=getTemperature(chip)
        waitHeating(10000)
```

Eine weitere Möglichkeit, die Einsatzbereitschaft zu testen, ist es, auf das Erreichen einer gewissen Gehäusetemperatur des MQ-3 zu warten. Aus der Tabelle, die ich mit dem Programm **heating**.py aufgenommen habe, erkennt man, dass bereits ab einer Gehäusetemperatur von 22°C keine Änderungen der Ausgangsspannung mehr auftreten. Die Funktion **awaitRunTemp**() misst fortlaufend die Temperatur und vergleicht diesen Wert mit einem fest eingestellten **tmin**, das ich zur Sicherheit in der Parameterliste am Programmbeginn auf 25°C gesetzt habe. Diese Temperatur ist beim Kaltstart nach gut 3,5 Minuten erreicht.

Zeit sec	Temp °C	U Volt
0	19,375	0,24
30	20,125	0,19469
60	21,25	0,183
90	22,125	0,181
120	23,065	0,18
150	23,8125	0,18
180	24,375	0,18
210	24,875	0,18
240	25,25	0,181

Dass es auch schneller geht, zeigt Spalte 3, **U Volt**. Hier wird die Eingangsspannung an A0 des ESP8266 aufgeführt und die ändert sich bereits nach 90 Sekunden ab Start nicht mehr signifikant. Das nutzt die Funktion **wait4Ustart**().

```
def wait4Ustart(uOld):
    sleep(5)
    U=getVal(50)[1]
    sleep(5)
    while abs(uOld-U) > 0.5:
        uOld=U
        U=getVal(50)[1]
        print(U)
        d.writeAt("WARTE AUF Ustart    ",0,1)
        d.writeAt("{0:2.3f} V       ".format(U),0,2)
        sleep(5)
```

Ich übergebe beim Aufruf zum Beispiel den Wert, den ich zuvor mit **getVal**() geholt habe, oder einfach nur 0. Dann wird erst einmal 5 Sekunden gewartet und ein neuer aktueller Spannungswert eingelesen. Solange der absolute Wert der Differenz aus **U** und **uOld** größer ist als 0,5, wird die Schleife durchlaufen. Aus neu wird alt, ein neuer Spannungswert wird geholt und ausgegeben, sowohl im Terminal als auch im OLED-Display. Fünf Sekunden warten und auf zur nächsten Runde.

wait4Ustart() kann zweierlei, die Routine kann das Vorheizen überwachen aber auch die Änderung der Spannung an RL verfolgen, wenn diese nach einem Test wieder langsam auf das Startniveau zurücksinkt. Erst dann sollte ein neuer Test stattfinden.

```
wait4Ustart(0)

Umax = 0
testende=TimeOut(testDauer)
d.clearFT(0,1)
d.writeAt("PUSTEN BEI GRUEN",1,1)
sleep(3)
green.on()
```

Ich warte auf das Erreichen der Startbedingung und stelle die Variable **Umax** auf 0. In der Schleife wir Umax die maximal erreichte Sensorspannung festhalten. Ich aktiviere einen Timer für das Ende des Testlaufs – Pusteanweisung aufs Display, drei Sekunden Pause und bei grün geht's los.

```
while 1:
    cnt,u=getVal(50)
    if u > Umax:
        Umax=u
    elif u < Umax and testende():
        break
    d.writeAt("{} cnt           ".format(int(cnt)),0,1)
    d.writeAt("{0:2.3f} mV       ".format(u),0,2)
    print(cnt,u)
    sleep(0.5)
green.off()
red.on()
d.clearFT(0,1)
conc=getConcentration(getRatio(getRS(Umax)))
print("Umax:", Umax, "   Concentration: {0:2.2f} \
Promille".format(conc))
d.writeAt("ATEM-ALK {0:2.2f} */**".format(conc),0,1)
d.writeAt("BLUT-ALK {0:2.2f} */**".format(2*conc),0,2)
```

Ein neuer Wert wird eingelesen und falls er das bisherige Maximum übersteigt, wird er als neues Maximum festgehalten. Sonst wird die Schleife verlassen, wenn auch noch der Timer **testende**() abgelaufen ist. Ich berechne aus **Umax** den Sensorwiderstand, damit das Verhältnis Rs/Ro und damit die Atemluftkonzentration. Die Ausgabe im Terminal und im OLED-Display schließen das Programm ab.

Ein Selbsttest mit drei Klaren hat ergeben, dass die Messung und Berechnung des Promillewerts mit 5% Abweichung nach unten gestimmt hat. Auch die Berechnung über Körpergewicht und Volumprozente ist ja nur eine Näherung. Daher übernehme ich keinerlei Garantie für die korrekte Arbeitsweise von Hard- und Software. Verlassen Sie sich also nicht darauf, dass das Ergebnis Ihres Nachbaus mit dem Ergebnis eines polizeilichen Alkomaten übereinstimmt. Am besten Sie handeln nach der Devise "Don't drink and drive"!