CO2-Sensoren zur Bekämpfung von Coronavirus-Aerosolen

Diese Anleitung ist aus einer Jugend forscht Arbeit entstanden und gibt einen Überblick über den Aufbau der Sensoren. Am Ende des Textes befinden sich alle zum Nachbau nötigen Links.

Einleitung

Eine Harvard-Studie[1] belegt, dass die Konzentrationsfähigkeit von Menschen bei erhöhter CO2-Konzentration in der Umgebung stark absinkt. Bei 400 PPM über dem Außenluftniveau verliert ein Mensch nach dieser Studie rund 21% seiner kognitiven Leistungsfähigkeit. Gerade in Klassenräumen können bei normalem Atmen der Schüler CO2-Konzentrationen auftreten, die die Leistungsfähigkeit des Gehirns stark verringern. CO_2 ist von Menschen nicht wahrzunehmen. Sie müssen also mithilfe von Geräten alarmiert werden. Bis heute waren solche Messgeräte für Schulen zu teuer und zu unflexibel. Messdaten konnten nicht zentral ausgewertet werden und die Alarmsysteme auch nicht an die Bedürfnisse der Schulen angepasst werden.

Meine rund 30 Euro teuren Sensoren ändern dies.

Mit einem WLAN-fähigen Arduino und einem günstigen NDIR-CO2-Sensor können Schüler und Lehrer per Blinklicht über eine zu hohe CO2-Konzentration benachrichtigt werden. Die Auswertung der gesammelten Daten ist cloudbasiert. Zudem besteht die Möglichkeit, die Sensoren in Schulprojekten nachzubauen. Auch die Effektivität der Rückmeldung konnte ich anhand eines Experiments nachweisen und damit zeigen, dass die Sensoren ihren Zweck voll und ganz erfüllen. Mein Projekt behandelt hierbei die Entwicklung der Hardware und Software der Sensoren, die Vorbereitung und Durchführung eines Schulprojekts zur Validierung der Nachbaufähigkeit der Sensoren und die Frage, inwiefern die Sensoren Menschen dazu animieren, zu lüften. Dies ist unter anderem nötig, um die durch das Coronavirus entstehenden Aerosole zu eliminieren.

Materialliste

Hier noch einmal eine vollständige Liste mit Materialien, die für den Bau, Betrieb und das Auslesen der CO_2-Sensoren notwendig war:

Software:

- Arduino IDE

- Thinkspeak Account

Werkzeuge für den Bau:

- Lötkolben/Zinn und sonstige Lötausrüstung
- Cutter

[1] Quellenverzeichnis: 2. Eintrag

- Computer mit USB-Anschluss und Netzwerkverbindung

Materialien für den Nachbau:

- MHZ19 Modul 0-5000ppm (20€)
- ESP8266 Microcontroller (3-6€)
- Female connector cable (0,5€)
- 3,3V-5V TTL Wandler (0,5€)
- Möglichst langes USB-Kabel (2€)
- Verteilerdose aus dem Baumarkt (1€)
- WS2812B LED-Streifen (1 Meter reicht für genau 5 Sensoren aus) 9€/5 -> ungefähr 2€ pro Sensor

Die Gesamtkosten belaufen sich theoretisch auf: 29 Euro

Hardwareentwicklung

Da ich mir zum Ziel gesetzt hatte, einen möglichst günstigen CO_2-Sensor zu bauen, der außerdem noch einfach anzupassen ist, war mir klar, dass ich die Arduino Plattform verwenden wollte. Mit dieser hatte ich schon gute Erfahrungen gemacht und es gibt eine gute Dokumentation, die es auch anderen Menschen erlaubt, den Code zu verstehen und zu verändern. Da die Daten der CO_2-Sensoren jedoch auch ausgewertet werden mussten, bot sich ein Controller an, der einen WLAN-Zugang hat. Die Auswertung ist zur Validierung der Nützlichkeit der Sensoren natürlich wesentlich. Aus eigener Erfahrung hatte ich den ESP8266 Microcontroller im Blick, der dem Benutzer erlaubt, sich an ein WLAN anzumelden und Daten per HTTP zu versenden.

Ein weiteres Teilziel war es auch, es Schülerinnen und Schülern möglich zu machen, die Sensoren selbst zu bauen. Dies hat den positiven Nebeneffekt, dass mit der Bekämpfung des Problems CO_2, Schülerinnen und Schülern auch das Thema Informatik und Technik nähergebracht wird. Dies brachte die Voraussetzung mit sich, beim Bau keine Werkzeuge zu verwenden, die nicht in einer Schule vorhanden sind.

Dabei orientierte ich mich an meiner eigenen Schule, die aufgrund ihrer altsprachlichen Orientierung einen guten Repräsentanten für die Mindestausstattung darstellt. Das heißt, dass davon auszugehen ist, dass Werkzeuge, die dort vorhanden sind, auch in den meisten anderen Schulen existieren.

Bei der Recherche nach dem CO_2-Sensor Modul selbst stieß ich bald auf den MHZ19 Sensor. Dies ist ein nichtdispersiver Infrarotsensor. Dieser funktioniert nach dem Prinzip, dass CO_2 eine bestimmte Wellenlänge (ungefähr 4300nm) infraroten Lichtes absorbiert. Das heißt, dass sobald mehr CO_2 in der Luft ist, weniger Lichtintensität bei dem im CO_2-Sensor eingebauten Infrarotempfänger ankommt. Dieser Effekt variiert auch nach Temperatur. Daher ist im Sensor auch noch ein Thermometer verbaut, was auch ausgelesen werden kann und die Nützlichkeit der CO_2-Sensoren weiter verstärkt. Hierbei kann auch der Effekt des stärkeren Lüftens auf die Temperatur im Innenraum gemessen werden. Außerdem muss der Sensor am Anfang kalibriert werden. Dazu muss man ihn für ein paar Minuten an die frische Luft legen und dann zwei Kontakte miteinander verbinden. Dann wird die Außenluft mit 400 ppm als Referenzwert gespeichert. Die Differenz zu den realen 410ppm ist nicht tragisch für unseren Anwendungszweck. Die Wahl fiel vor allem auf dieses Modell, da es gegenüber chemischen Modellen, wie dem MG811, keine lange Aufwärmzeit hat und vor allem nur auf das Gas

CO_2 reagiert. Das MG811 ist ein Modul, das eine Heizspannung braucht und zum Aufheizen rund 24 Stunden benötigt. Daher ist es nicht dafür geeignet, spontan angeschaltet zu werden.

Der MHZ19 kann die Daten per serieller Übertragung an den Mikrocontroller senden.

Zu guter Letzt, war es nötig, dem Sensor eine Möglichkeit zu geben, aktiv zu melden, wann zu viel CO_2 in der Innenluft vorhanden ist. Da die Sensoren von der Schulgemeinde jedoch angenommen werden müssen, war es jedoch nötig, eine Art der Rückmeldung zu finden, die subtil genug ist, dass sie nicht nervte und die Schulgemeinde nicht dazu brachte, die Sensoren abzuschalten. Dabei sollte sie trotzdem klar genug sein, dass sie die Schulgemeinde dazu motivierte, zu lüften. Damit schied die akustische Rückmeldung aus der engeren Auswahl aus, da sie bei den befragten Lehrerinnen und Lehrern nicht auf Begeisterung stieß. Es blieb die visuelle Rückmeldung. Damit man diese möglichst flexibel programmieren kann, entschied ich mich dazu, RGB-LEDs zu verwenden. Um aussagekräftige Animationen zu gestalten, habe ich dann LED-Streifen verwendet. Diese geben die Möglichkeit, LEDs mit geringem Lötaufwand, einzeln anzusteuern. Die WS2812B Streifen eigneten sich dazu sehr gut, da man sie auch mit einer Haushaltsschere verkürzen kann und damit genau an die eigenen Ansprüche anpassen kann. Ein LED-Streifen reicht hierbei für genau 5 Sensoren.

Da einige Komponenten auf 5 Volt arbeiten und andere auf 3,3 Volt, habe ich unter anderem auch noch einen TTL-Wandler verbaut. Dieser kann vier Ein- und Ausgänge ansteuern. Vorteilhaft ist, dass dieses Modul sehr günstig ist. Die 5 Volt Referenzspannung erhält das Modul vom Mikrocontroller selbst, der die Eingangsspannung vom USB-Anschluss ungeregelt weitergibt. Von dort muss dann diese Spannung an den CO_2-Sensor, den Referenzeingang des TTL-Wandlers und an den VIN-Pin des LED-Streifens gegeben werden.

Als Gehäuse dient eine Verteilerdose aus dem Baumarkt, die mit einem Cutter aufgeschnitten werden kann, um Löcher für Kabel und Luftzufuhr zu schaffen.

Als Stromversorgung lässt sich jedes USB-Netzteil verwenden. In unserer Schule hatte es sich jedoch als besonders praktisch erwiesen, dass Smartboards mit einem freien USB-Port vorhanden waren. Diese schalten sich mit dem Smartboard ein und auch wieder aus. Damit ist gewährleistet, dass während es Betriebs immer eine Person anwesend ist. Ein normales USB zu micro-USB Kabel kann hier verwendet werden.

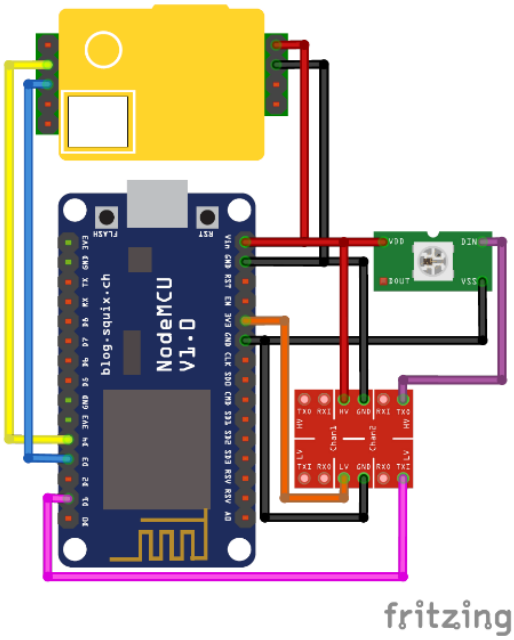
Um eine Übersicht zu gewinnen, habe ich einen Schaltplan mit der freien Software „fritzing“[2] erstellt. Diesen habe ich möglichst bildlich gehalten, um ihn später im Schulprojekt mit den Schülerinnen und Schülern verwenden zu können.

Der Aufbau ist hier in einer Schritt-für-Schritt Anleitung kurz dargestellt und ist sonst dem Schaltplan und den eingefügten Bildern zu entnehmen:

1. Anlöten der Pins an alle Platinen
2. Verbinden der Pins nach dem Schaltplan
3. Schneiden eines passenden Lochs für den CO_2-Sensor in die Verteilerdose
4. Einkleben der Komponenten und Anbringen des selbstklebenden LED-Streifens auf der Außenkante
5. Aufspielen der Software mit vorher einprogrammierten WLAN-Anmeldedaten

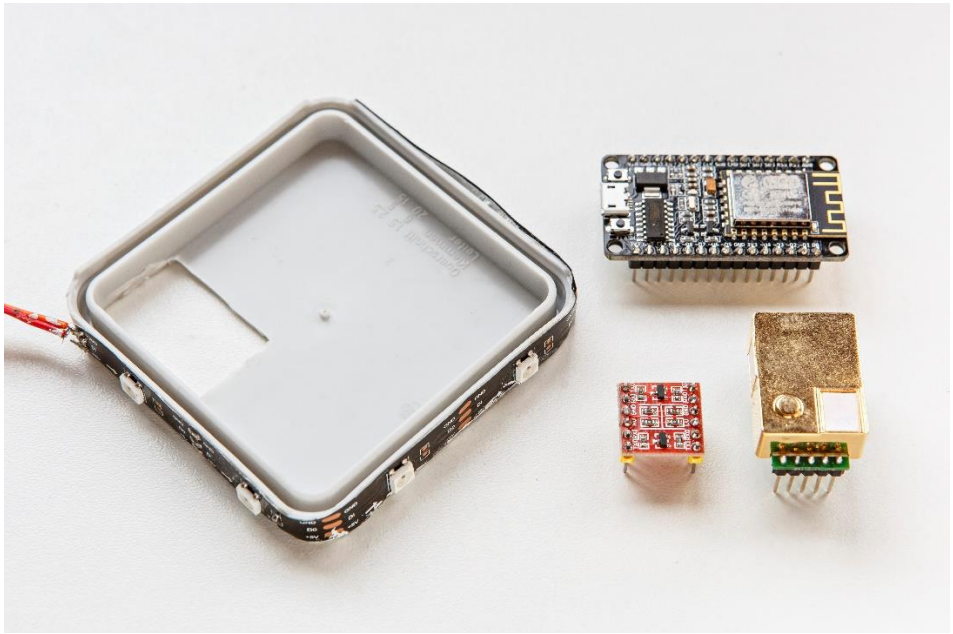
[2] Quellenverzeichnis: 6 c). Eintrag

Schaltplan:



Die LED-Leiste, der Sensor und der TTL-Wandler werden mit 5 Volt aus dem VIN-Pin des ESP8266 verbunden (Rot). Das gleiche gilt für GND (Schwarz). Orange stellt hier die 3,3 Volt Referenzspannung für den TTL-Wandler (Rote Platine) dar. Die rosa markierte Leitung wird im TTL-Wandler auf die benötigten 5 Volt geregelt und verlässt diesen als violettes Kabel. Die LED-Leiste funktioniert nur mit 5 Volt Signal und Versorgungsspannung. Das gelbe Kabel stellt die RX Datenleitung dar, wobei das blaue Kabel die TX Datenleitung darstellt.

Einzelteile:



Hierbei sind die Bauteile: Der ESP8266 (oben rechts), der MHZ19 (unten rechts), der TTL-Wandler (unten mittig), die LED-Leiste (links)

4.2. Software

Um den Microcontroller zu programmieren, braucht man die Arduino-Umgebung (IDE)[3]. Diese lässt sich auf jedem PC mit einem der drei gängigen Betriebssysteme installieren und nutzen.

[3] Quellenverzeichnis: 6 a). Eintrag

Die Anforderungen, die die Software erfüllen musste, habe ich in einer Liste zusammengetragen:

- Einfachheit und Anpassbarkeit
- Daten an einen Server zu senden, der diese speichern und wenn möglich auswerten kann
- Rückmeldung der von schlechter Luft in einfacher Form an die Schulgemeinde
- Die Möglichkeit zu bieten, die Software per WiFi zu updaten
- Sich mit einem WLAN zu verbinden

Nach kurzer Recherche fand ich den IoT-Dienst „Thingspeak“[4] [5], bei dem man in einer kostenlosen, abgespeckten Version die Möglichkeit hat, alle 15 Sekunden Daten an einen sogenannten Channel zu schicken. In diesem Channel werden die Daten dann gespeichert und können einfach als .csv Datei exportiert werden. Außerdem gibt es auch eine rudimentäre Visualisierung der Daten. Die Daten werden zeitlich zugeordnet abgespeichert. Des Weiteren besteht auch die Möglichkeit, die Daten per MATLAB zu verarbeiten. Diese Channel sind über eine Website öffentlich abrufbar und lassen es zu, die Daten auch von Schülerinnen und Schülern einsehen zu lassen. Es ist bei der kostenlosen Version maximal möglich, vier Sensoren anzuschließen, weswegen man schnell an Grenzen gelangt. Es ist jedoch auch möglich, die Thingspeak Software auf einem eigenen Server bereitzustellen und damit alle Regulationen zu umgehen.

Für diesen Dienst existiert auch eine Bibliothek[6], mit deren Hilfe es dem Endanwender einfach gemacht wurde, die Daten zu verschicken. Die zugehörigen Befehle ließen sich aus der Dokumentation herleiten.

Auch für den CO_2-Sensor gibt es eine Bibliothek[7], die aber leider unfertig war. Deshalb musste ich einige Dateinamen umbenennen. Nachdem ich den Code und die Dokumentation des CO_2-Sensors gelesen hatte, funktionierte diese jedoch ohne Probleme. Gewisse Befehle (zum Beispiel der Check, ob der Sensor schon aufgewärmt war (mhz19.isWarming())) funktionierten nicht, ließen sich aber umgehen. Denn: nach einigem Testen stellte sich heraus, dass der Sensor immer den Wert „410“ sendet, solang er nicht aufgewärmt ist.

Die LED-Leiste lässt sich über die Bibliothek von Adafruit[8] ansteuern und dient dazu, die Schulgemeinde auf einfache Art und Weise auf schlechte Luft aufmerksam zu machen. Die programmierten Animationen sollten zu jedem Zeitpunkt laufen, weswegen es nicht möglich war, nach jedem Senden der Daten, einfach 15 Sekunden zu warten. Eine Animation wird aktiviert, wenn der zugehörige Schwellenwert erreicht ist. Dabei habe ich mich entschieden, einfache, klare Animationen einzusetzen: Ein leuchtendes Rot aller LEDs als Warnfarbe soll die Schulgemeinde aufmerksam werden lassen. Ein blinkendes Rot verstärkt dies noch. Alle Animationen, die einen Status anzeigen, sind grün und zeigen damit die Bereitschaft des Systems. Wenn der CO_2-Konzentration im akzeptablen Bereich ist, zeigt der Sensor keine Farbe an, sondern blinkt alle 60 Sekunden kurz grün, um den Status anzuzeigen.

Des Weiteren nutzte ich die Bibliothek ArduinoOTA.h[9], die dem Anwender die Möglichkeit gibt, die Software über WiFi zu updaten. Dabei muss sich der PC im gleichen WiFi-Netzwerk befinden wie der Arduino. Diese Bibliothek war mir aus früherer Erfahrung schon bekannt. Dabei ist es auch möglich, ein Passwort zu setzen, was vor unbefugtem Zugriff schützt. Für die Bibliothek ist es nötig, Python 2.7 auf dem Rechner zu installieren, der das Programm sendet. Leider ist die Funktion, die das Update

[4] Quellenverzeichnis: 6 b). Eintrag
[5] Quellenverzeichnis: 8. Eintrag
[6] Quellenverzeichnis: 5 e). Eintrag
[7] Quellenverzeichnis: 5 a). Eintrag
[8] Quellenverzeichnis: 5 b). Eintrag
[9] Quellenverzeichnis: 5 c). Eintrag

initiiert, zeitkritisch. Man muss sie periodisch aufrufen, weswegen man nie zu lang warten darf, bevor man sie erneut aufruft.

Außerdem musste noch die Wifi-Verbindung initialisiert werden, was sich als sehr schwierig herausstellte. Denn der ESP8266 lässt sich nur sehr schwer mit Enterprise-Wifis verbinden, die an Schulen häufig zu finden sind. Der Hersteller des Mikrocontrollers „Espressif" hat hier seine API noch nicht vollständig ausgearbeitet und wird es wohl auch nicht mehr tun. Nach vielen Stunden des Lesens der offiziellen Dokumentation der API und mehreren Fehlschlägen wandte ich mich an meinen Physiklehrer/Netzwerkadministrator, der die Möglichkeit hatte, ein WPA2-PSK Netzwerk zu erstellen. Mit diesem war es möglich, den Sensor mit dem WLAN zu verbinden[10]. Es stellte sich außerdem heraus, dass der nächstbessere Controller (ESP32) einfacher mit Enterprise-Netzwerken zu verbinden ist. Dieses Problem verschlang einige Wochen an Zeit.

Meine Vorgehensweise bestand daher vor allem darin, eine Software zu gestalten, die ohne Pausen auskommt und damit in Echtzeit reagieren kann. Kein Prozess sollte sich behindern. Dies ist wichtig, da Microcontroller immer nur einen Prozess gleichzeitig abarbeiten können. Daher war die oberste Prämisse nach einer Aktion, nicht zu warten, sondern für jede Aktion zu überprüfen, ob „die richtige Zeit" gekommen war. Dies bedingt sich durch die Limitationen, die durch die Animationen, die OTA-Bibliothek (Over the Air) und den Thingspeak-Server gegeben waren.

Um Übersicht zu bewahren, habe ich mehrere Funktionen deklariert.

- In check(); wird zuerst geprüft, ob das Intervall von 16 Sekunden vergangen ist, sodass neue Daten gesendet werden können. Diese werden dann ausgelesen und an Thingspeak verschickt. Außerdem wird der Timer wieder auf 0 gesetzt, sodass weitere 16 Sekunden gewartet werden können, bis wieder Daten an Thingspeak gesendet werden können. 16 Sekunden waren nötig, da Thingspeak das erneute Senden von Daten auf minimal 15 Sekunden limitiert.
- In Wifi(); wird das Wifi initialisiert. Dabei hatte ich zuerst in Betracht gezogen, das Schulische Enterprise-Wifi zu nutzen, doch später WPA2-PSK (personal shared key) genutzt. Diese Funktion wird nur einmal aufgerufen und ist daher nicht zeitkritisch. Wird jedoch kein WiFi gefunden, bleibt der Controller hier stecken. Es muss noch eine Möglichkeit gefunden werden, einen Timeout zu definieren, der die Funktion des Sensors auch bei fehlendem Netzwerk gewährleistet. Trotzdem muss gewartet werden, da der Controller eine kurze Zeit braucht, um sich mit dem Netzwerk zu verbinden.
- In loop(); werden die Animationen ausgeführt, die alle darauf basieren, möglichst keine Zeit zu verbrauchen. Das heißt: Bei jeder Aktion wird geprüft, ob die Animation gerade verändert werden soll oder nicht. Die Animation wird mit einem pixels.show(); geupdatet. Vorher werden alle Animationsdaten der Bibliothek übergeben. Zu den Animationen gehören:
    - Eine blaue Status LED, die alle 60 Sekunden einmal blinkt
    - Ein rotes Leuchten aller LEDs, wenn der CO_2-Konzentration über einen bestimmten ersten Schwellenwert steigt.
    - Ein rotes Blinken, wenn der CO_2-Konzentration über einen bestimmten zweiten, höheren Schwellenwert steigt.
    - Ein grünes Blinken, wenn der Sensor angeschaltet wird. Dies geschieht, um zu signalisieren, dass der Sensor bereit ist, zu messen.

[10] Quellenverzeichnis: 5 d). Eintrag

Ein grünes Leuchten war angedacht, um den Schülerinnen und Schülern mitzuteilen, dass der CO_2-Konzentration genug gesunken ist, wurde jedoch von der Lehrerschaft als zu ablenkend empfunden und daher nicht umgesetzt.

Ein delay(); von 10 Millisekunden wird aufgerufen, da eine höhere Aktualisierungsrate nicht gefordert ist und der Prozessor nicht unnötig laufen muss.

Nachbau der Sensoren in einem selbstgeleiteten Schulprojekt

Um nachzuweisen, dass die Sensoren einfach nachzubauen sind, habe ich ein Schulprojekt veranstaltet, in dem 5 Schülerinnen und Schüler die Sensoren nachgebaut haben. Die Elternspende unserer Schule hat die Kosten, die sich für 5 Sensoren auf rund 170 Euro beliefen, dankenswerterweise zur Verfügung gestellt. Die Bauteile habe ich bei Banggood, einem internationalen Online-Händler, bestellt. Sie sind jedoch auch mittlerweile (September 2020) in Deutschland für einen geringeren Preis zu erhalten.

Um die Schülerinnen und Schüler vorzubereiten, habe ich allen den Schaltplan zur Verfügung gestellt. Des Weiteren hatten sie die Möglichkeit, einfache Lötaufgaben zu üben, indem sie zuallererst eine Platine mit allen mit 5 Volt belegten Kabeln verbanden. Diese konnte nicht zerstört werden und hatte damit einen guten Übungswert. Außerdem habe ich noch die Pins vorher an die CO_2-Sensoren gelötet. Dies verhinderte die Zerstörung dieser. Nach drei Projekttagen mit jeweils 4 Stunden Zeit waren die Sensoren fertig und wurden sogar von ein paar besonders kreativen Schülerinnen bemalt. Es war zu keinem Zeitpunkt die Hilfe einer Lehrkraft vonnöten. Während dieser Zeit konnte ich mich auch um die letzten Fehler der Software kümmern. Wichtig ist in Schulen, dass die Arduino-Treiber auf den Rechnern installiert sind. Ich musste mir mit einem Laptop abhelfen, da ich keinen Admin-Zugang besaß. Zu guter Letzt war es notwendig, die Software auf die Sensoren zu spielen und die MHZ19-Module zu kalibrieren. Dies funktionierte alles ohne Probleme. Das Endresultat bestand aus 5 funktionierenden CO_2-Sensoren.

Links:

a. Arduino: https://www.arduino.cc/ (Abrufdatum: 14.01.20)
b. Thingspeak: https://thingspeak.com/ (Abrufdatum: 14.01.20)
c. Fritzing: https://fritzing.org/home/ (Abrufdatum: 14.01.20)
d. MHZ19B: https://github.com/crisap94/MHZ19 (Abrufdatum: 14.01.20)
e. Adafruit Neopixel: https://github.com/adafruit/Adafruit_NeoPixel (Abrufdatum: 14.01.20)
f. ArduinoOTA: https://arduino-esp8266.readthedocs.io/en/latest/ota_updates/readme.html (Abrufdatum: 14.01.20)
g. Arduino Wifi: Ist in Arduino enthalten. Daher: https://www.arduino.cc/ (Abrufdatum: 14.01.20)
h. Thingspeak: https://github.com/mathworks/thingspeak-arduino
i. Installation des ESP8266: http://arduino.esp8266.com/stable/package_esp8266com_index.json (Abrufdatum: 14.01.20)
j. all mein Code für den ESP8266, https://github.com/Jasper-Sic/CO2-Sensor (Abrufdatum: 14.01.20)