CO2Duino

Make your own CO2 meter

Michiel Ryvers

itenium 2022

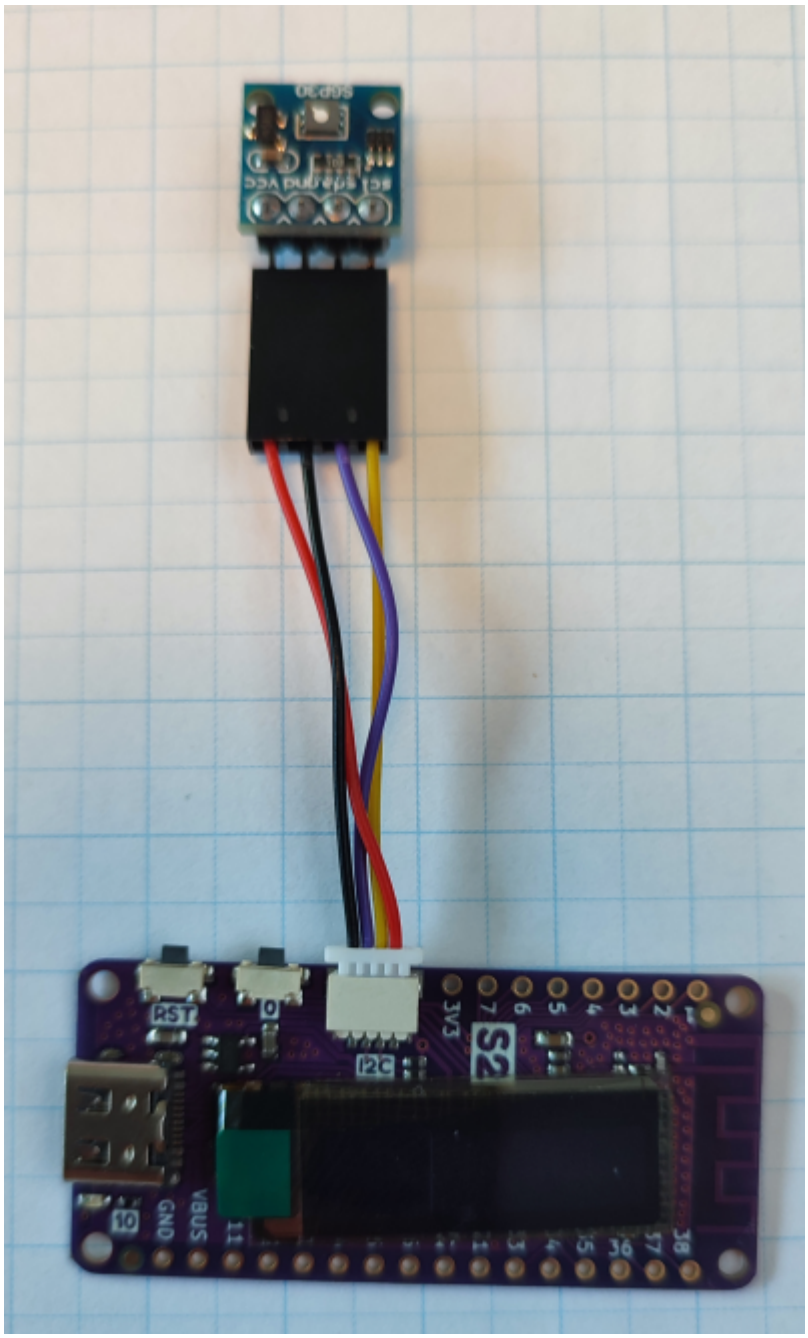
Table of contents

1. CO2duino

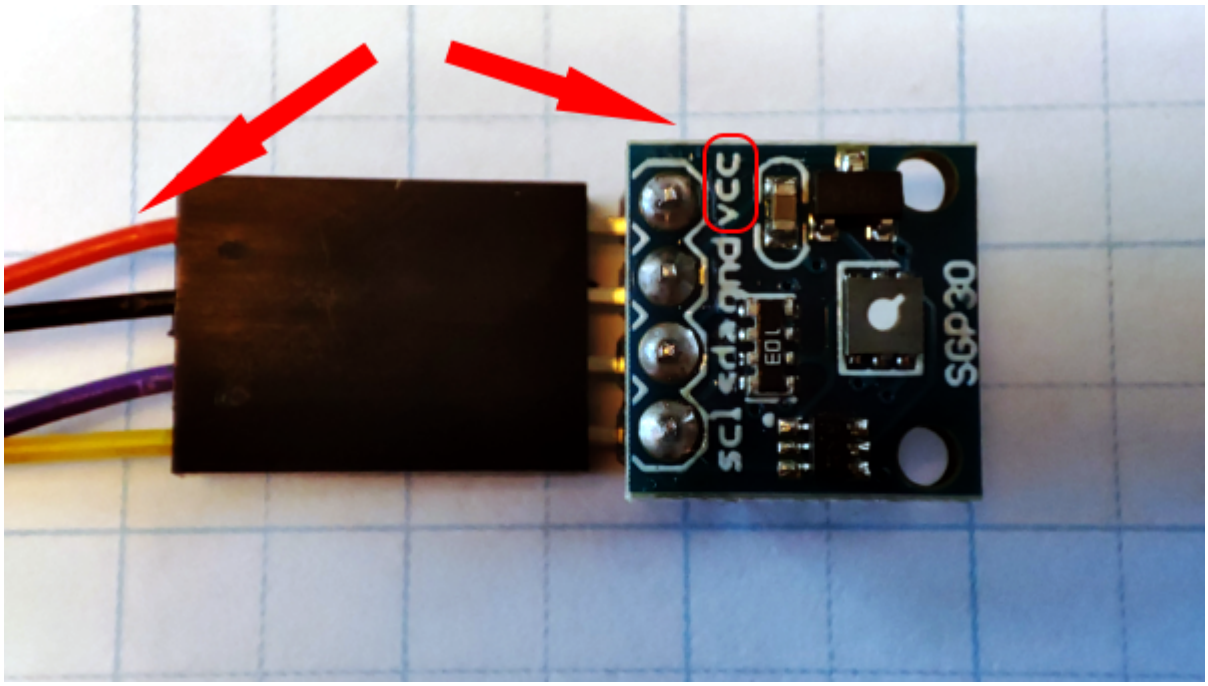
A DIY CO2 meter based on the Wemos S2 Pico dev board and a SGP30 CO2 sensor. It's programmed using PlatformIO and the Arduino framework. This information is also available online.

1.1 Initial hardware setup

To get started you'll have to connect the sensor to your development board with the provided cable. The small white connector plugs in to the development board and the black connector goes to the CO2 sensor.



⚠ **Note** Pay special attention to the sensor connection! The red wire should go to the "VCC" pin on the sensor. See this image.



Assembly of the board and sensor into the enclosure is described in the Assembly guide, this is not necessary to get started programming.

1.2 Initial software setup

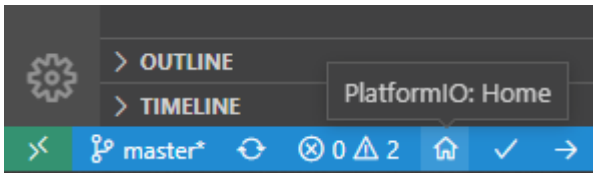
⚠ **Note** We use Visual Studio Code for this project, install it first if it isn't already.

First we'll clone this repository:

```
git clone https://gitlab.com/michielr/co2duino.git
```

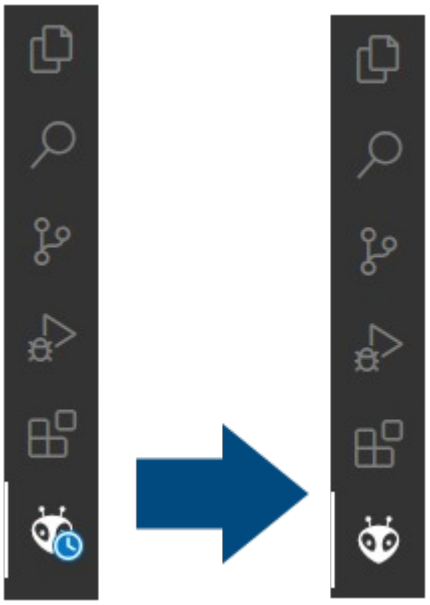
Next we have to install the PlatformIO extension into Visual Studio Code. Go to extensions, search for `PlatformIO` and press install. The initial install takes some time, you'll get notified after about 2 minutes to reload your VS Code window.

After VS Code has reloaded you should see an icon of a house in the bottom of your VS Code window:



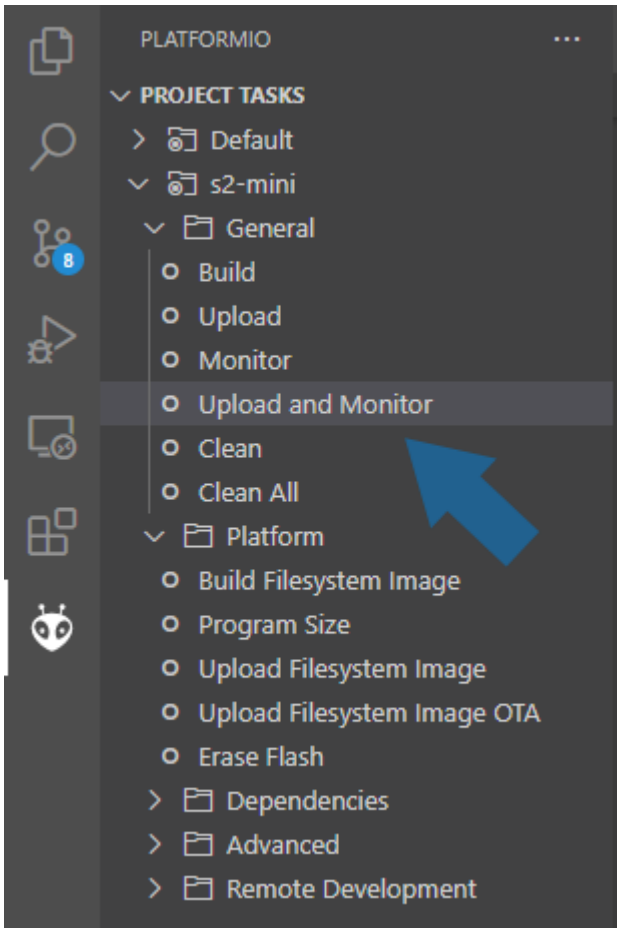
Click on this icon and you'll be greeted by the PlatformIO Home screen. On this screen select 'Open Project' and navigate to your cloned repository to open it.

After pressing `Open`, PlatformIO will begin installing all the dependencies of the project. This will take another couple of minutes. You'll know when it's done when the clock icon is gone from the PlatformIO tab.



Now connect your device to your computer, open `src/main.cpp` and have a first look at the code. (Maybe change some displayed text?)

Now we're going to upload our code to the device for the first time! Open the PlatformIO tab (see above) and click on the "Upload and monitor" task.



A terminal will pop open on the bottom of your screen and PlatformIO will start compiling your code and then upload it to the connected board. If all goes well, you'll see a green `=== SUCCESS ===` message and the device will reboot itself running your code! PlatformIO should also have opened the "Serial monitor", these are messages the device is sending to your computer. It should print `Hello from the Serial connection!` every second.

You're all set to get started with the exercises!

1.3 Exercises

In these exercises we'll build up to a fully functioning CO2 meter, the exercises aren't fleshed out but provide some reading material to get started. As a starting point the program (found in `src/main.cpp`) shows a bootscreen and some text on the display.

1. Initialize the CO2 sensor and read data from it
2. Display sensor data on the screen
3. Save the calibration data to the device
4. Connect to a known WiFi network
5. Set up an access point so the user can choose their WiFi network and enter its password
6. Send a notification to your phone using IFTTT
7. Log data to ThingSpeak
8. Make it pretty
9. Configure URLs
10. Further improvements

1.4 Resources

1.4.1 Enclosure

- Assembly guide

1.4.2 Final code

- `master` branch of this repository

1.4.3 CO2 Sensor (Sensirion SGP30)

- SGP30 Datasheet
- Adafruit SGP30 library

1.4.4 Display

- Adafruit SSD1306 OLED driver
- Adafruit Graphics library

1.4.5 WiFi connectivity

- WiFiManager

1.4.6 Utility

- Preferences

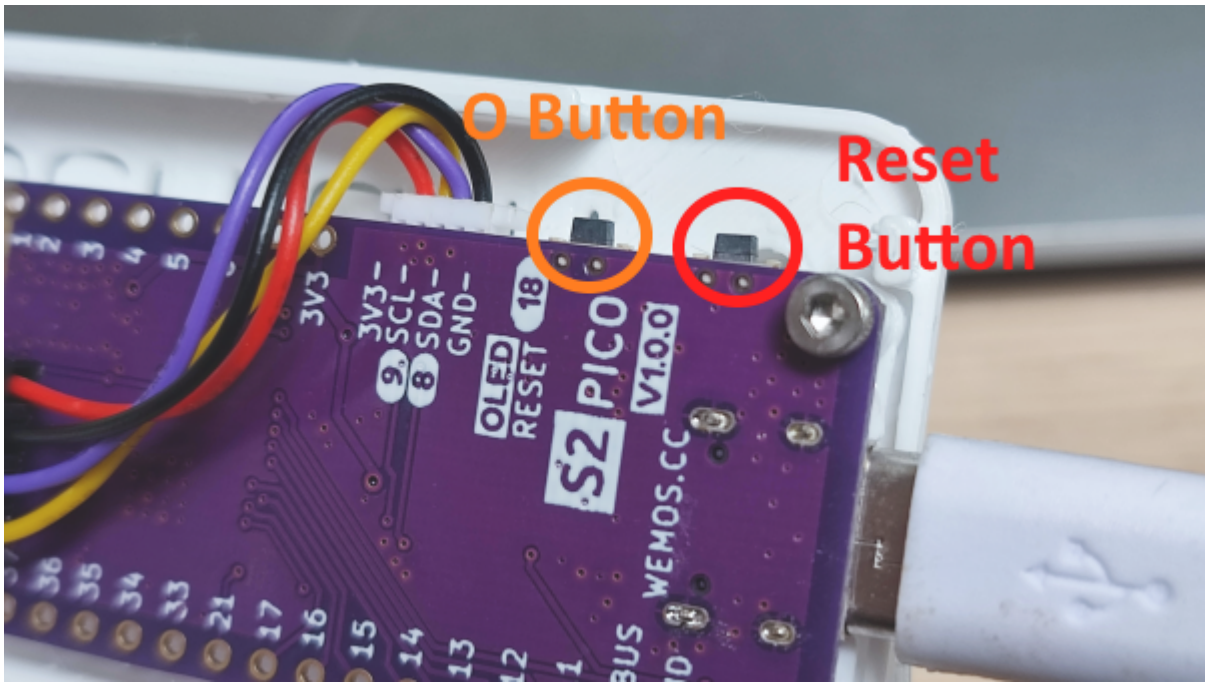
1.4.7 Tutorials

- Random Nerd Tutorials
- Adafruit Learn
- Savjee tutorials and blog

1.5 Troubleshooting

1.5.1 My code won't flash! There is no COM port found!

No worries, we can manually set the device to "Flashing mode" by pressing the two tiny buttons on the side of the board at the same time. Your device will reboot and be ready to flash again. You might have to manually reset the board after flashing by pressing the tiny "Reset" button.



1.5.2 Upload and monitor asks me for a COM port

PlatformIO might be starting the monitoring task a bit too fast. Try to use the "Upload" task instead of "Upload and monitor" and when you've seen your device reboot open the "Monitor" task.

2. Exercises

2.1 Initializing the sensor

Check the Adafruit SGP30 github repo to read about the sensor we're going to use.

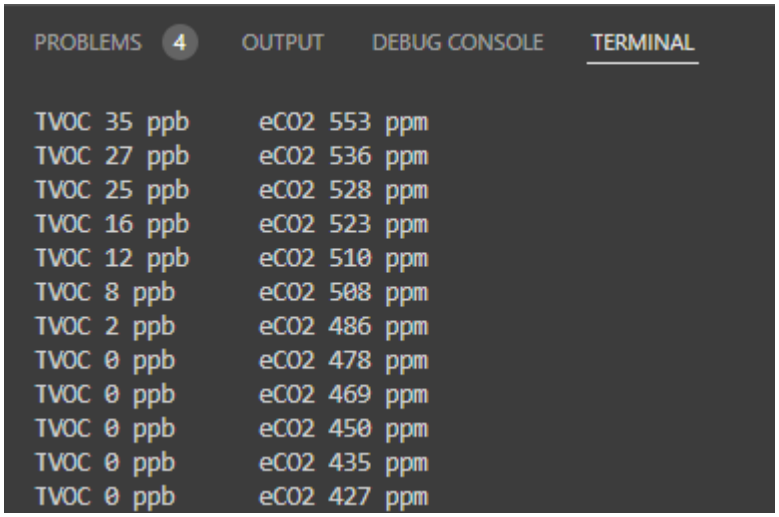
The library is already installed in the project, so check out the provided example in the GitHub repository and start coding!

Set up the sensor in the `setup()` function and read it out in the `loop()` function. Write out the data each second to the serial monitor using `Serial.println(data)` and a `delay(1000)` at the end of your loop.

When building and uploading your program to the microcontroller, make sure to use the `Upload and Monitor` option in the PlatformIO Project tasks. This will automatically open the serial monitor after the program is flashed.

⚠ **Note** if the sensor only outputs a value of 400 for the CO2 and 0 for the TVOC, don't panic, this is normal in the first ~30s of polling.

2.1.1 Expected result



Everything works? Great! Head on over to the next exercise, displaying this data on the display!

2.2 Displaying the sensor data

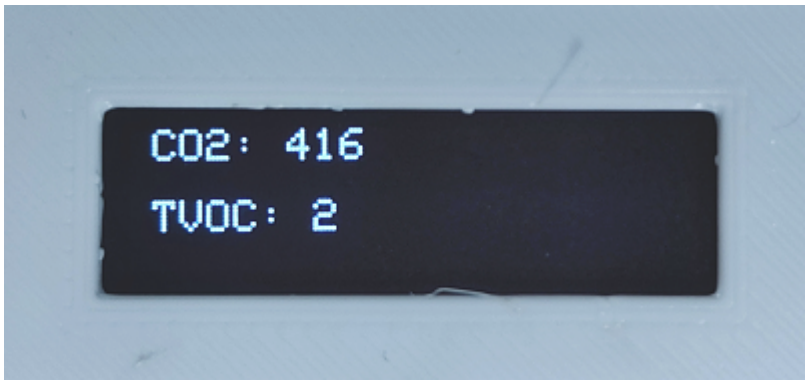
Okay, so we've got our sensor sending data to our computer over its `Serial` connection, now let's try displaying this data on the OLED display!

Your device should still be blinking "Hello" at you, play around with this to display the captured data instead.

2.2.1 Useful links

- Random Nerd Tutorial
- Adafruit SSD1306 OLED driver
- Adafruit Graphics library

2.2.2 Expected result



Everything works? Great! Head on over to the next exercise, Save the calibration data to the device!

2.3 Sensor calibration

The sensors we use get calibrated in the factory, but to accurately measure CO2 values it needs a baseline. On earth the concentration of CO2 in the outside air is around 400 ppm. So the sensor expects that the lowest value it has read should be around this value (the baseline). You can read this "baseline measurement" from the chip and save it somewhere so the next time you start the sensor you can tell it what its last baseline measurement was. To get the baseline measurement, have a look at the provided example in our SGP30 library.

We'll save this data each minute to some storage on our microcontroller that doesn't get erased when we power cycle the device. We'll use the Preferences library to do this. See the linked tutorial below.

To bring it all together, in your `setup()` function, check if we've got some saved baseline measurements. If so, set these measurements after initializing the sensor.

In the `loop()` function we should check if a minute has passed. If so, get the baseline measurements and save them using the Preferences library.

⚠ **Note** Not everything will go as planned on the first try, make it a habit to send some debugging data to the `Serial` line to debug your code when it is running on the microcontroller.

2.3.1 Useful links

- Random Nerd Preferences tutorial
- Adafruit SGP30 library

2.3.2 Expected result

```
--- Available filters and text transformations: colorize, debug, defaul
--- More details at https://bit.ly/pio-monitor-filters
--- Miniterm on COM8  115200,8,N,1 ---
--- Quit: Ctrl+C | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H ---
Got values from Preferences:
TVOC_base: 37199
eCO2_base: 37864
Setting previously stored baseline values
Saving baseline values to EEPROM
TVOC_base: 37199
eCO2_base: 37864
```

Everything works? Great! Head on over to the next exercise, Connect to a known WiFi network!

2.4 Connect to a known WiFi network

In this exercise we'll connect to a known WiFi network, check out the provided example and make sure your program prints out its IP address after connecting to the WiFi network.

2.4.1 Useful links

- WiFi Client basic example

2.4.2 Expected result

```
--- Available filters and text transformations: colorize, debug, default, direct,
--- More details at https://bit.ly/pio-monitor-filters
--- Miniterm on COM14  115200,8,N,1 ---
--- Quit: Ctrl+C | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H ---
.
WiFi connected
IP address:
192.168.0.138
```

Everything works? Great! Head on over to the next exercise, Set up an access point so the user can choose their WiFi network and enter its password!

2.5 Set up an access point

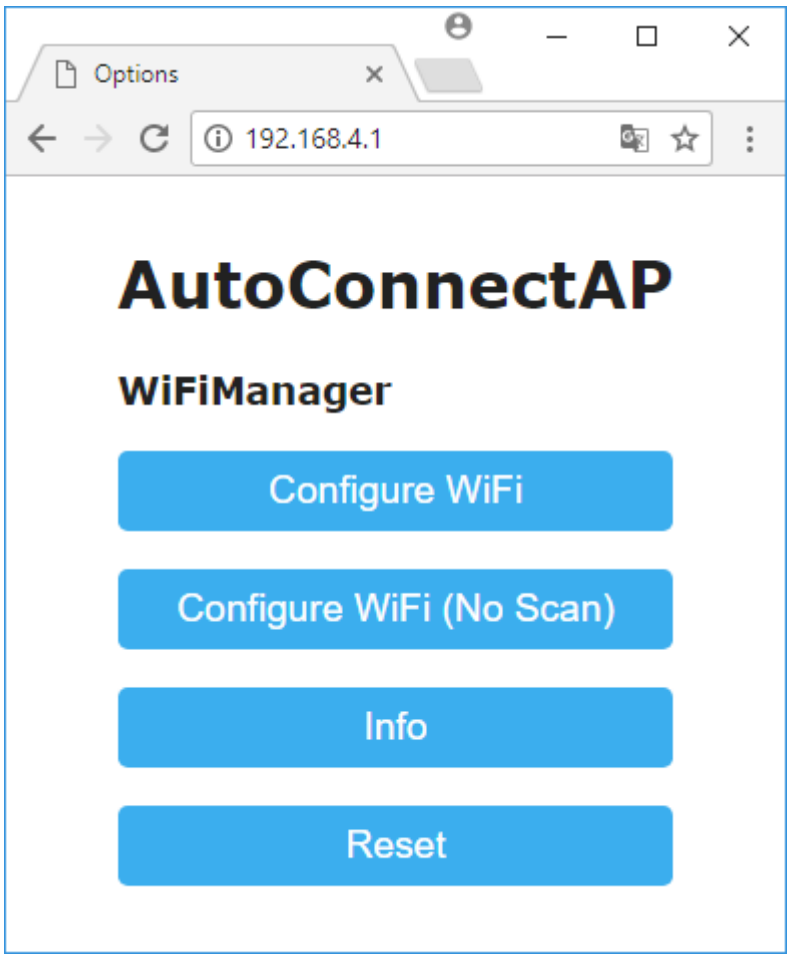
Since we want to be able to move our CO2 meter, we can't always hardcode our WiFi network into the program.

WiFiManager is an Arduino library we can use to allow the user to connect to the CO2 meter and then select their home WiFi network. Check out the provided tutorial and github link to let the user select their own WiFi network when the CO2 meter starts up.

2.5.1 Useful links

- Random Nerd WiFiManager Tutorial
- WiFiManager library (outdated docs)

2.5.2 Expected result



Everything works? Great! Head on over to the next exercise, Send a notification to your phone using IFTTT!

2.6 Send a notification to your phone using IFTTT

A high concentration of CO2 in the air is detrimental to a lot of things, we have a harder time concentrating, diseases get spread more efficiently and it can lead to headaches. To remind us to open up a window once in a while we'll set up an IFTTT action that sends us a notification whenever the measured CO2 in the air is above 800. Check out the tutorial linked below to see how to do this. Make sure you only send a notification when the last one is at least 10 minutes ago.

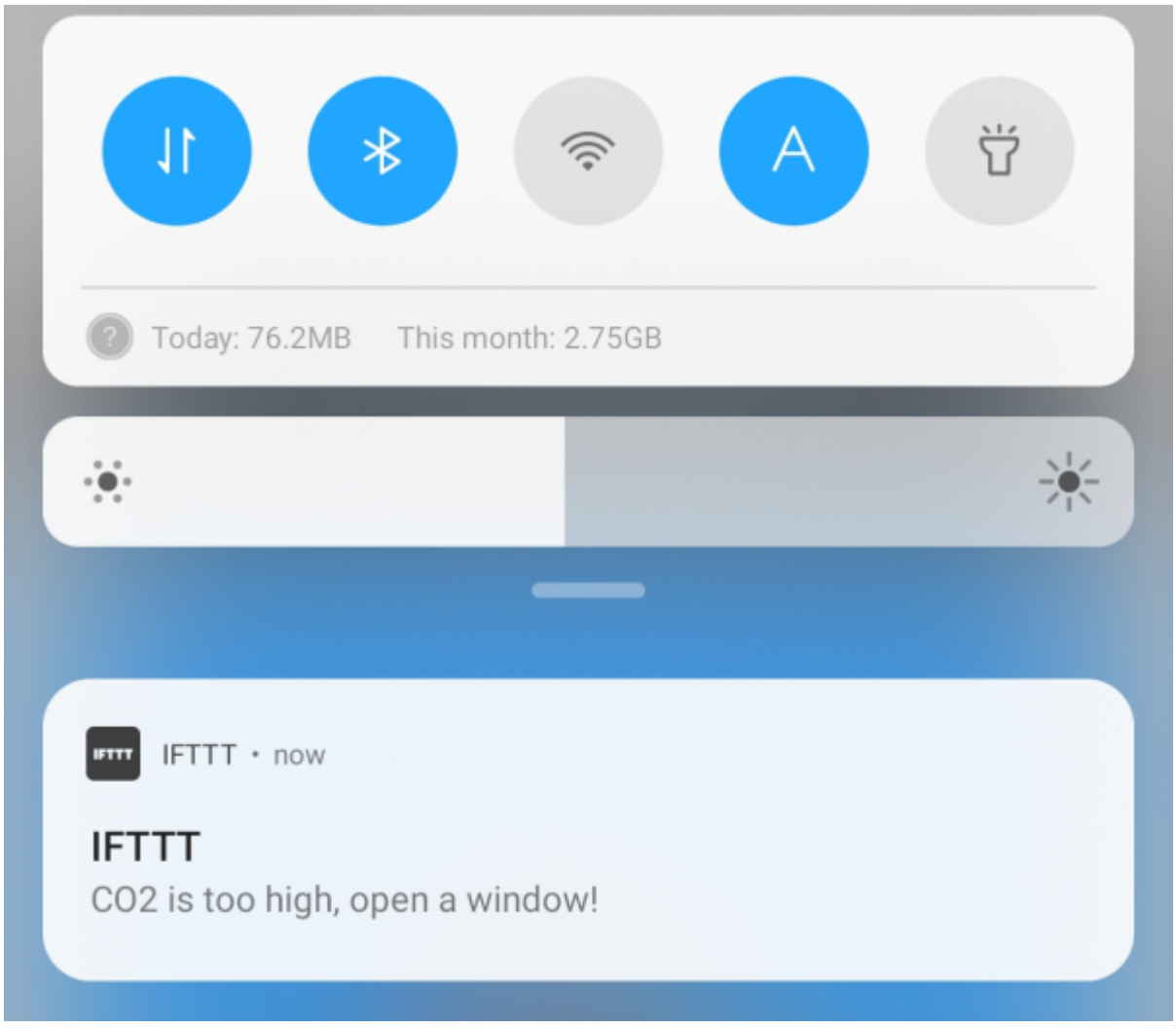
⚠ **Note** To receive notifications from IFTTT on your phone you have to install their app.

⚠ **Note** A hardcoded IFTTT URL is fine for the moment, we'll make it configurable in a next exercise

2.6.1 Useful links

- ESP32 - IFTTT tutorial
- Arduino millis() method, see how much time has passed

2.6.2 Expected result



Everything works? Great! Head on over to the next exercise, Log data to ThingSpeak!

2.7 Log data to ThingSpeak

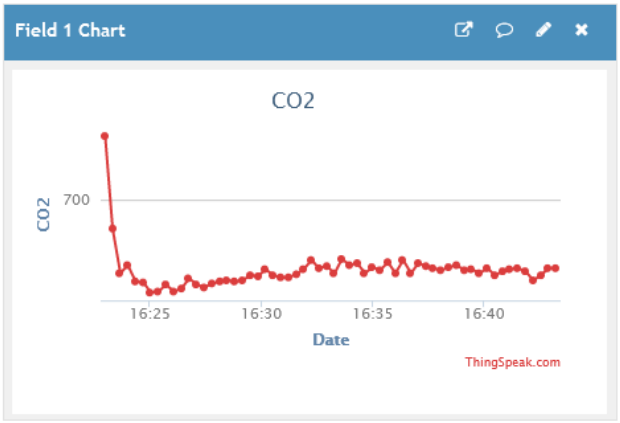
To get an historical view of our data (maybe to see some trends) we need to log our data. In this example we'll use ThingSpeak as a host to log our sensor data. They provide some nice graphs and a generous free tier. Check out the tutorial below and log your CO2 and TVOC data every 10 seconds to ThingSpeak.

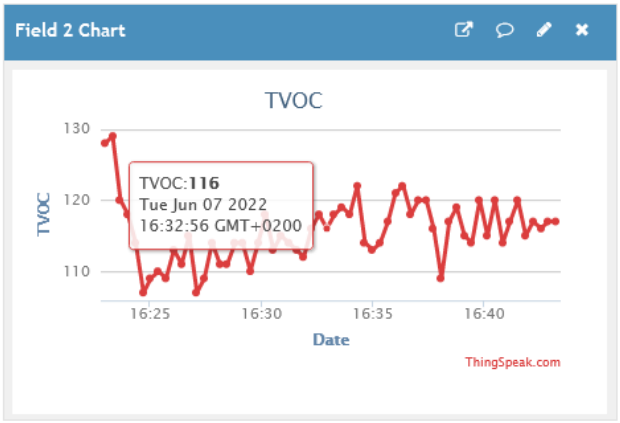
⚠ **Note** A hardcoded ThingSpeak URL is fine for the moment, we'll make it configurable in a next exercise

2.7.1 Useful links

- Random Nerd ThingSpeak Tutorial

2.7.2 Expected result





Everything works? Great! Head on over to the next exercise, Make it pretty!

2.8 Make it pretty

In this exercise we'll try to make our displayed result a bit prettier. Check out the display libraries' documentation below and make it your own!

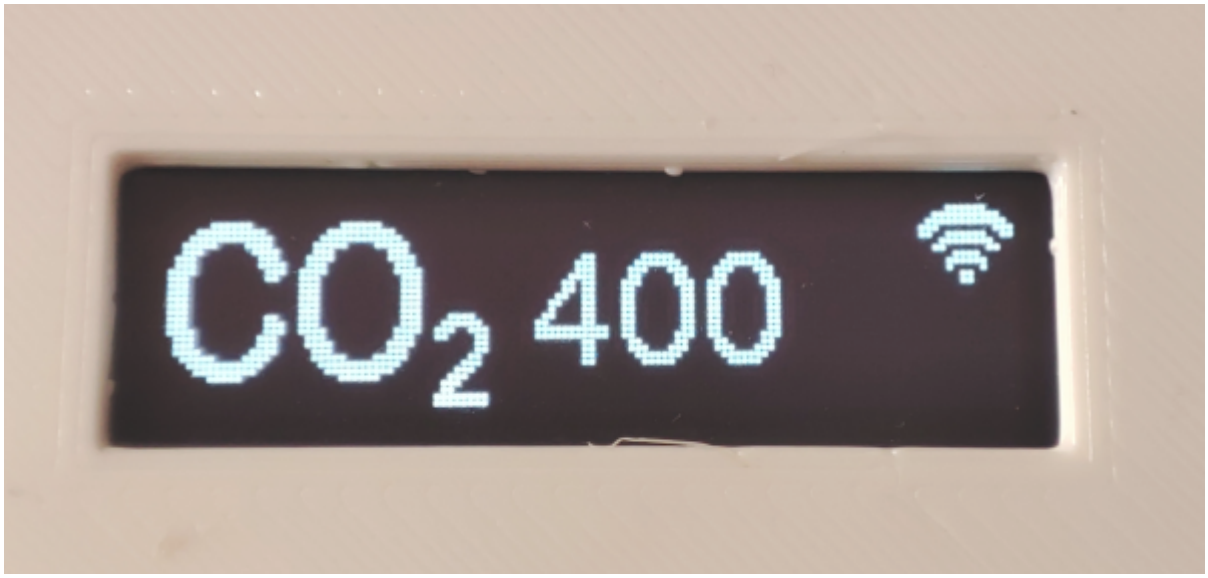
Some ideas:

- Try to use the icons provided in the `icons.h` file
- Let the user know if the device is connected to the WiFi with an icon
- Blink a warning sign if the measured CO2 is over 800
- Try out some fonts!

2.8.1 Useful links

- Adafruit SSD1306 OLED driver
- Adafruit Graphics library

2.8.2 Example result



Everything works? Great! Head on over to the next exercise, Configure URLs!

2.9 Configure URL's

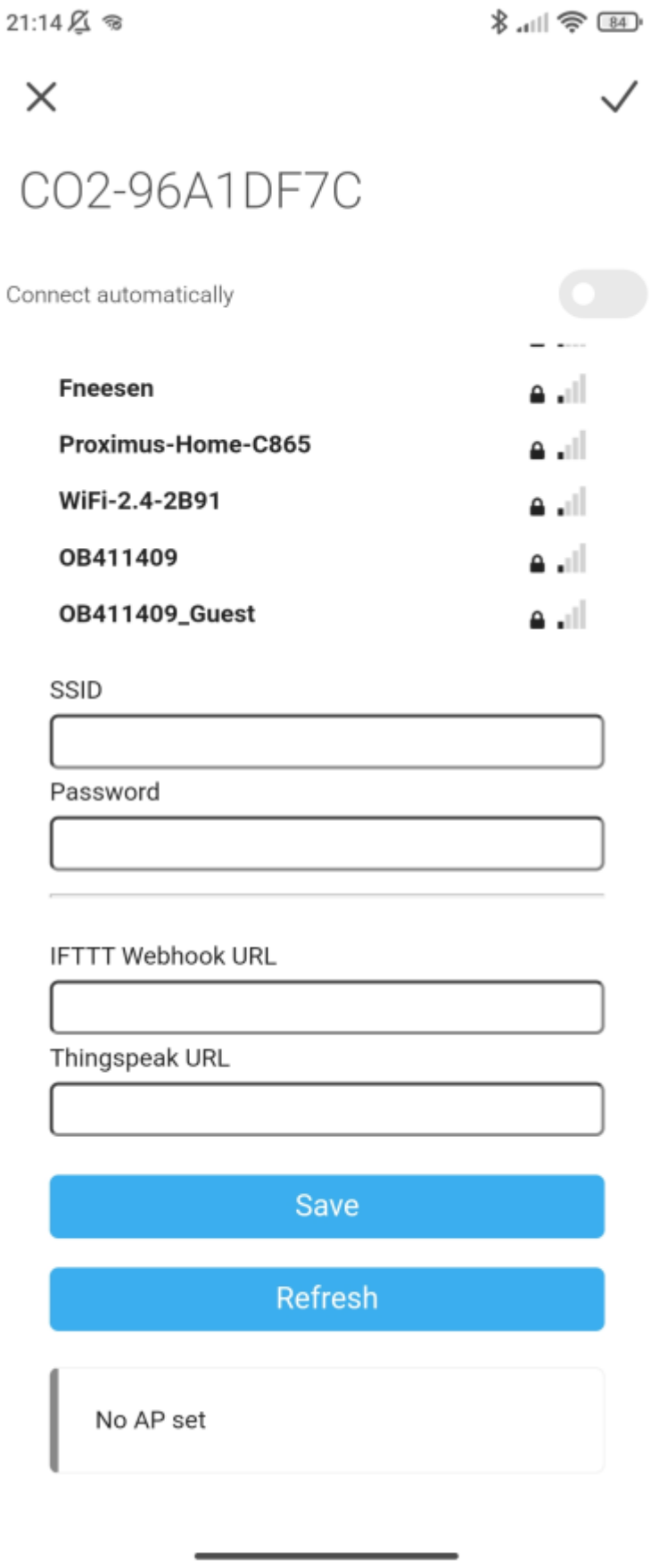
We still have two hardcoded URL's in our project, the ThingSpeak URL and the IFTTT URL. In order to make these configurable we'll use parameters in the WiFiManager project. The user can then provide these URL's when they are connecting to their own WiFi. Read the provided Random Nerd WiFiManager Tutorial to get started!

Set the provided URL's to a global variable and save them using the Preferences library for the next boot! (See the sensor calibration exercise for a refresher).

2.9.1 Useful links

- Random Nerd WiFiManager Tutorial
- Random Nerd Preferences tutorial

2.9.2 Expected result



Everything works? Congratulations! You've built a feature complete CO2 sensor! Give yourself a high five!

Hungry for more? See the Further improvements exercise where I've provided you with some reading material and ideas to implement some advanced features!

2.10 Further improvements

Wow, you've built a complete CO2 meter with smart capabilities, all from scratch! I'm super proud of you!

If this workshop has scratched an itch for you and you'd like to go a bit further down the rabbit hole I've listed some ideas and resources here to get you started.

Some of them are implemented in the final firmware you can find on the `master` branch of this repository. So don't hesitate to have a look!

2.10.1 Tasks and threading

Did you notice your device seemed to hang when it was sending an HTTP request? This is because all the code we've written runs one after another. It might have even felt a bit iffy while writing it. To fix this problem our microcontroller runs a RTOS (real time OS). This gives us many possibilities to fix this problem. Try to offload the sending of notifications and logging of data to a separate task. Check out the following materials:

- Multitasking on an ESP32 with Arduino and FreeRTOS
- Introduction to FreeRTOS Video tutorials

2.10.2 Mutexes and Semaphores

What if we put everything the device does into separate tasks? Reading data, updating the display, saving the calibration data, ... We might run into problems when for example the "Read data" task and the "Save calibration data" task both try to access the CO2 sensor at the same time. This is where mutexes and semaphores come in. Try to request access to the sensor in both tasks using a mutex:

- Introduction to FreeRTOS: Mutexes
- Introduction to FreeRTOS Github

2.10.3 Code style

Up to now, most of our code exists in one huge `main.cpp` file. Read up on C++ code organization to make everything tidy.

- Source File Organization for C++ blog post

2.10.4 Monitor WiFi connection

What if our WiFi password changes? The device has no way of reconnecting or notifying the user something is wrong. Write a task to see if our connection is still alive and allows the user to reconnect or reconfigure the WiFi.

- Keep WiFi alive using a FreeRTOS task

2.10.5 Watchdog

What if our device hangs on a specific function? Maybe a reboot would get everything running smoothly again. This is where Watchdogs come in. Implement a Task Watchdog that monitors for long running tasks and reboots the device if something is taking far too long.

- ESP32 Task watchdog documentation

2.10.6 Change parameters at runtime

What if we'd like to change the URL the notification gets sent to without resetting the device or rewriting a part of our program? Maybe the device listens on it's USB Serial port for a message to change it? Maybe it can spin up a web server where you can update this information? Get creative!

- Listening to USB serial on a web page
- ESP32 Web server

3. How to assemble the CO2 meter

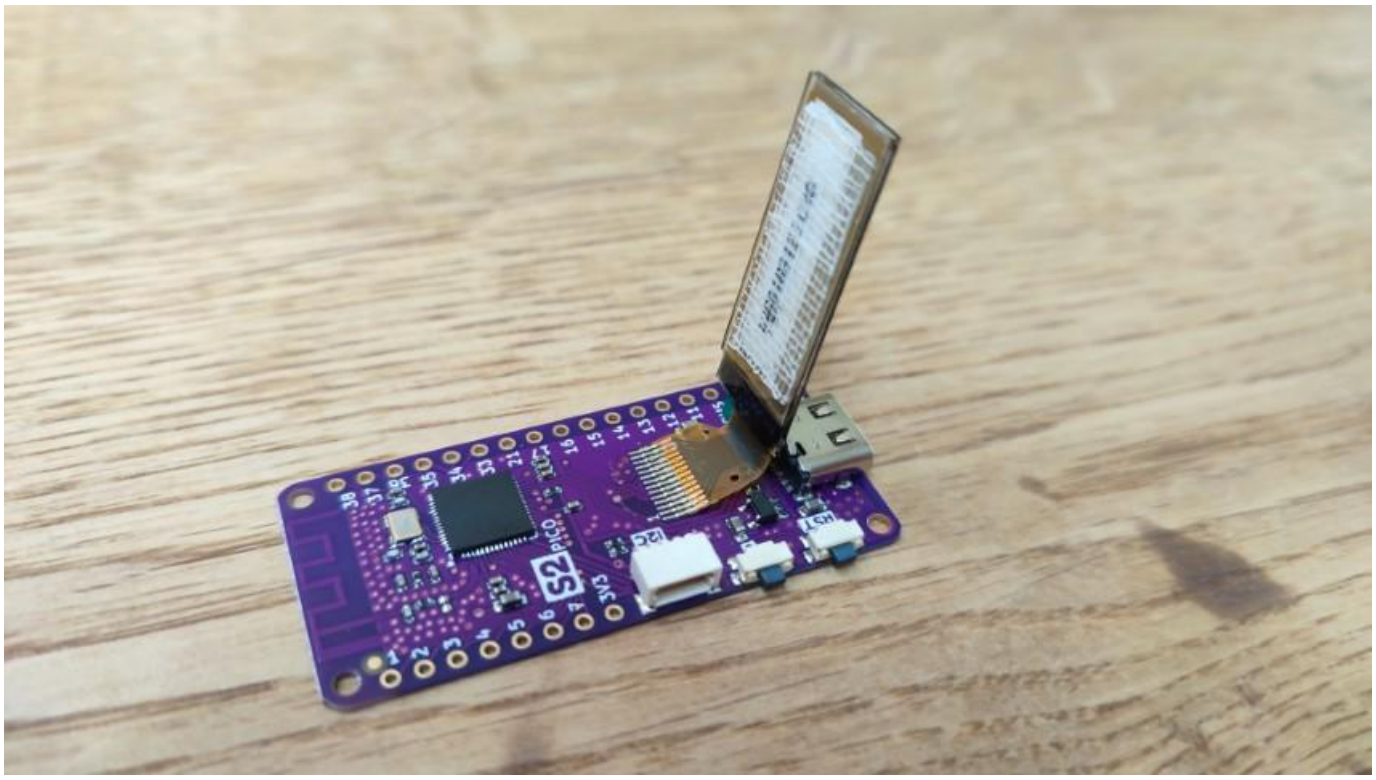
3.1 Gather materials

Gather the required materials and components, you'll need:

- Wemos S2 Pico board
- SGP30 Sensor
- Connecting cable
- Enclosure (3 parts)
- 3 screws
- Screwdriver

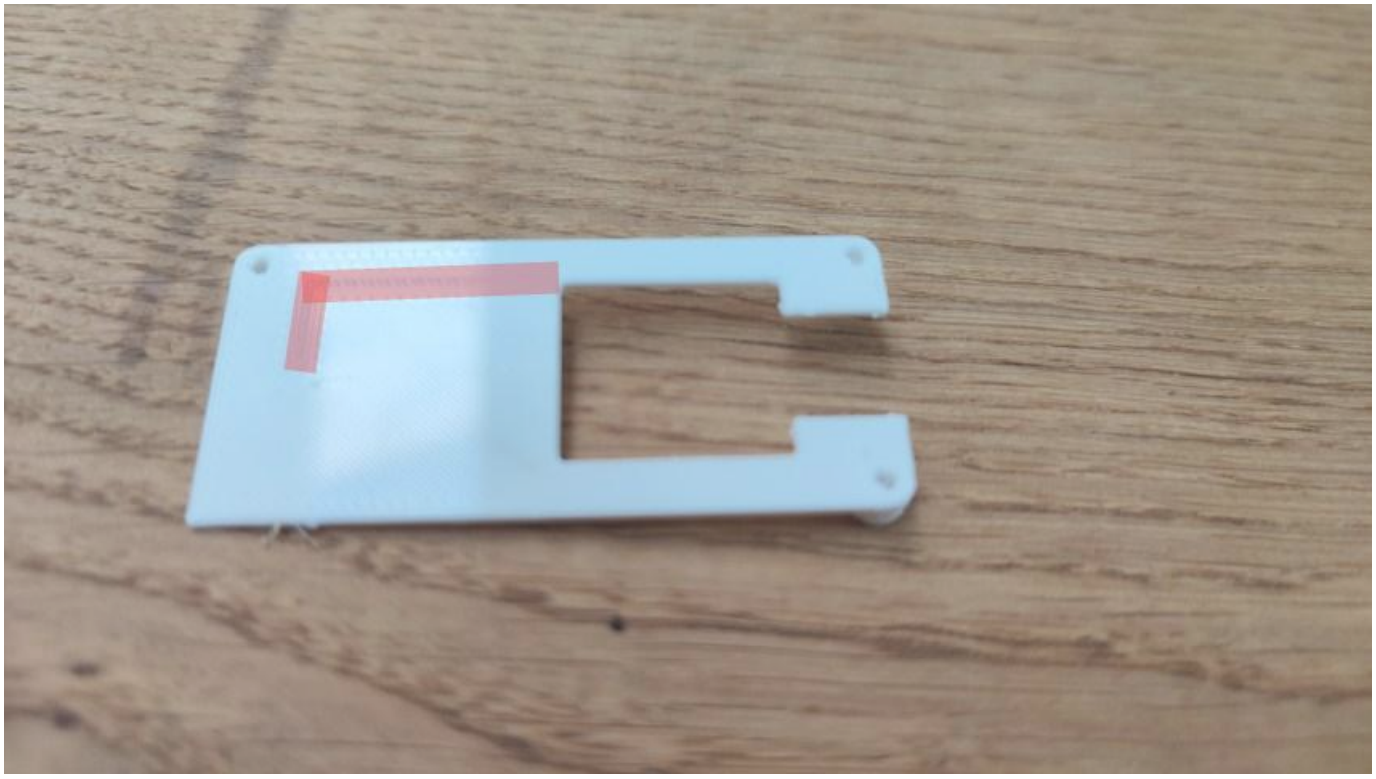
3.2 Release OLED from board

Slowly peel up the OLED display from the board, taking care to leave the adhesive material on the OLED, not the board.



3.3 Paste OLED on board spacer

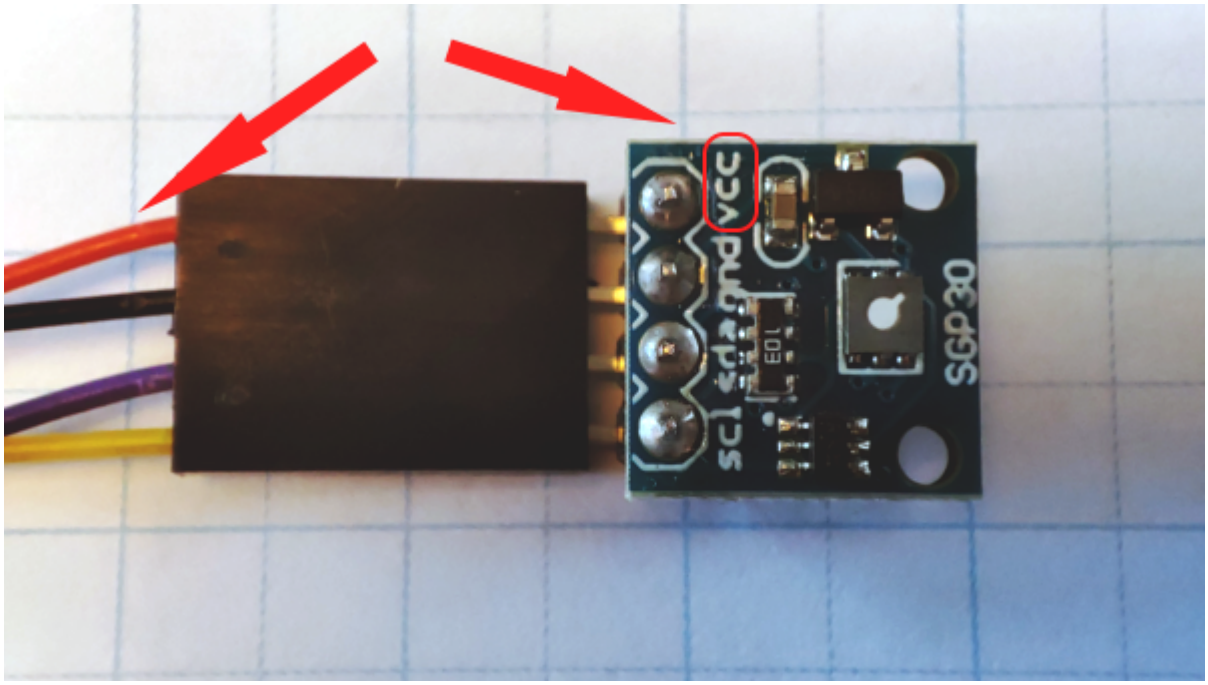
First, notice the small indentation on the spacer, this is where you'll align the screen. Marked in red in the next picture.

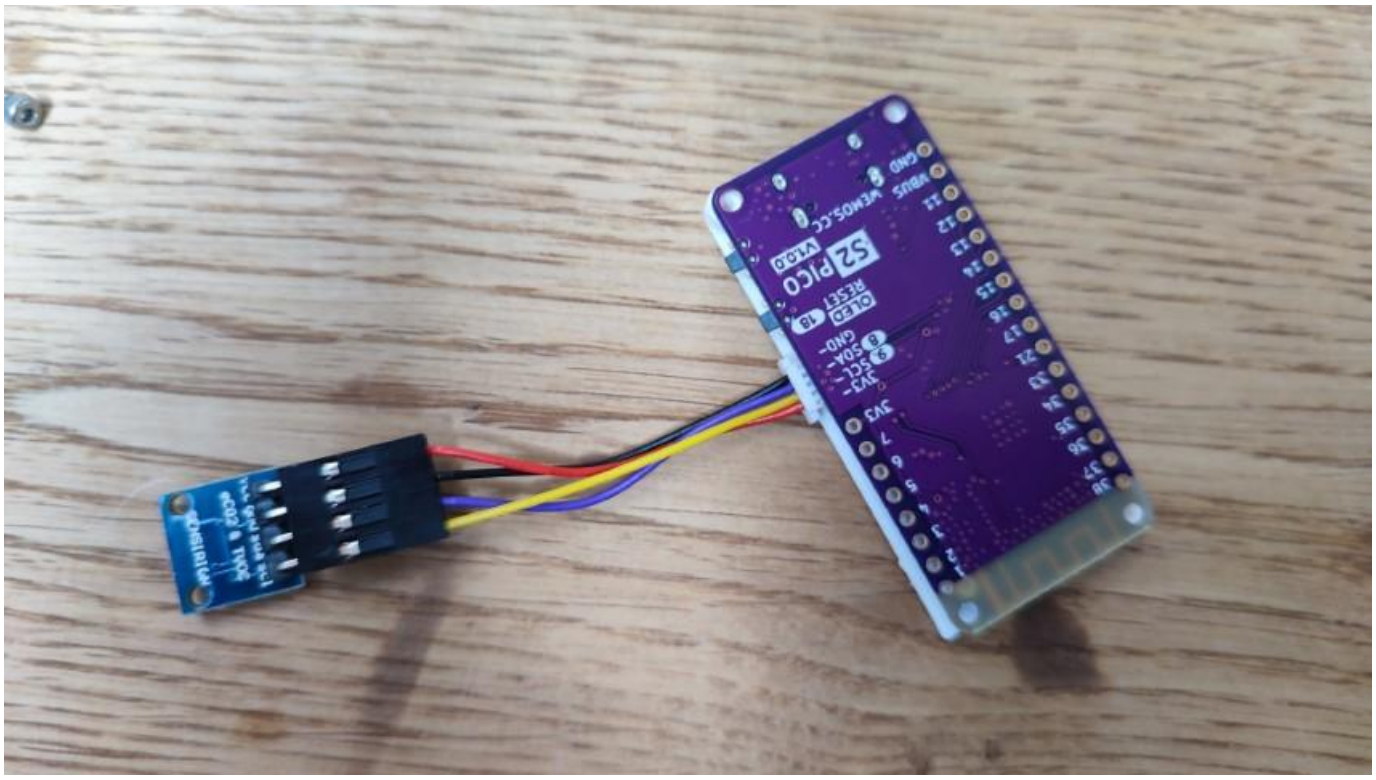


Next align the OLED with the spacer and press it down. Don't forget to remove the OLED plastic cover after this step. (I sure didn't, ahum)

3.4 Connect the sensor to the board

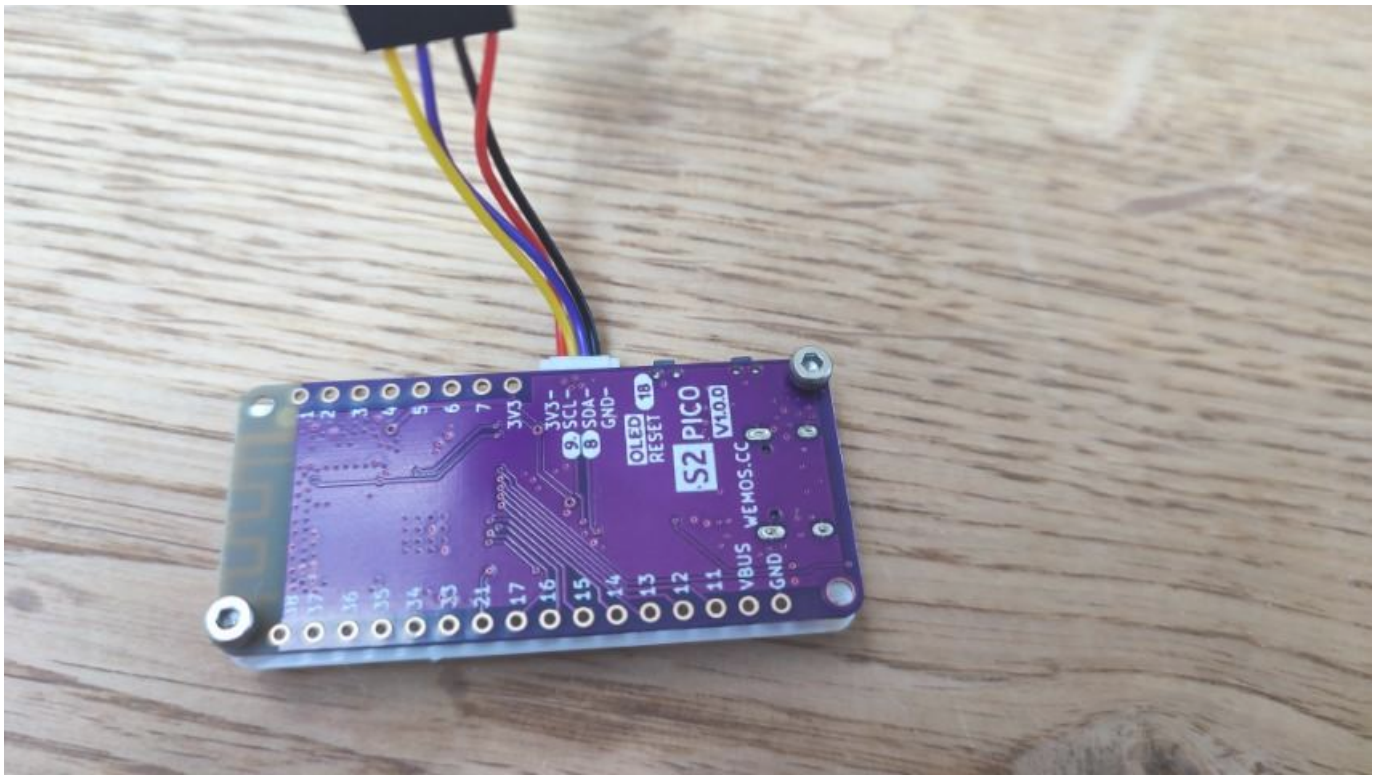
Next we'll connect the sensor to the board. Pay special attention to the sensor connection! The red wire should go to the "VCC" pin on the sensor. See this image.



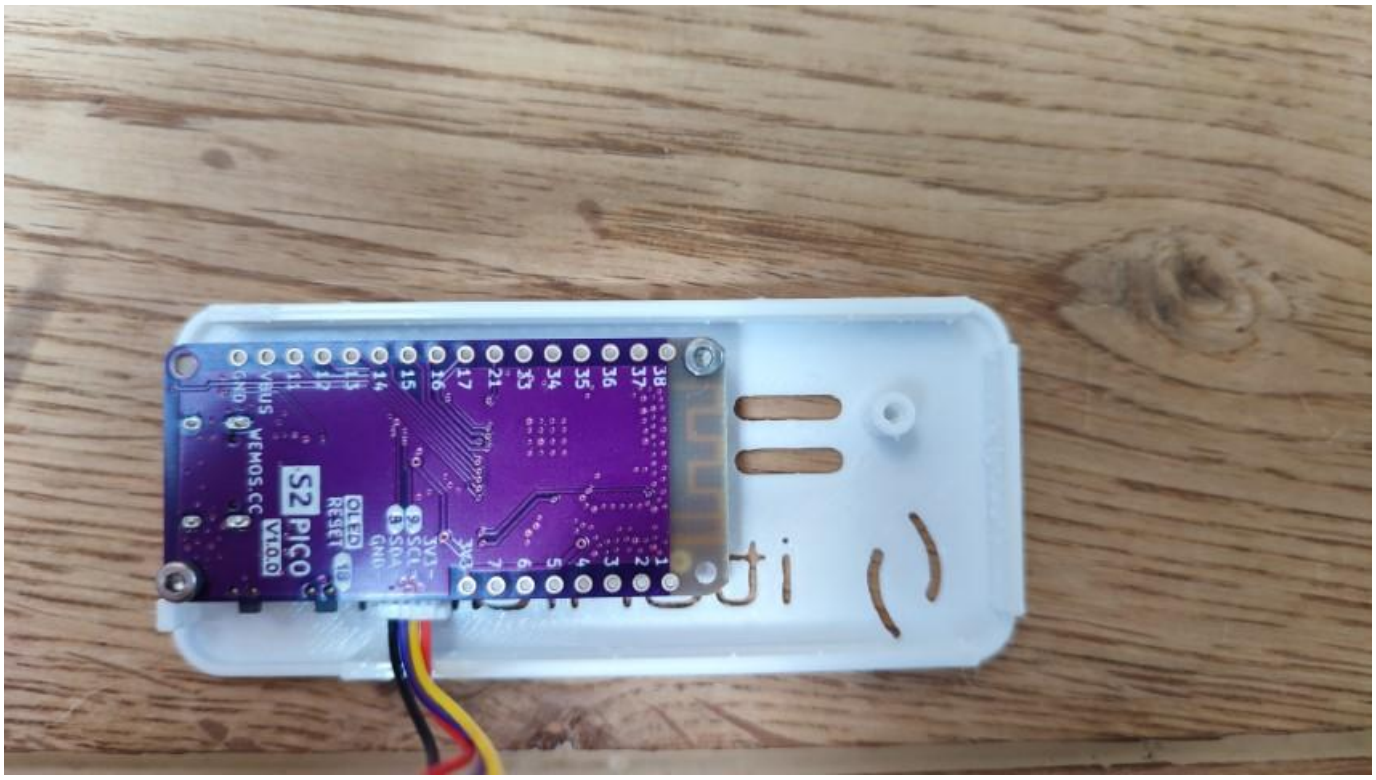


3.5 Mount the board in the enclosure

Start the screws that go from the board into the spacer part until you can just see the head of the screw coming out.

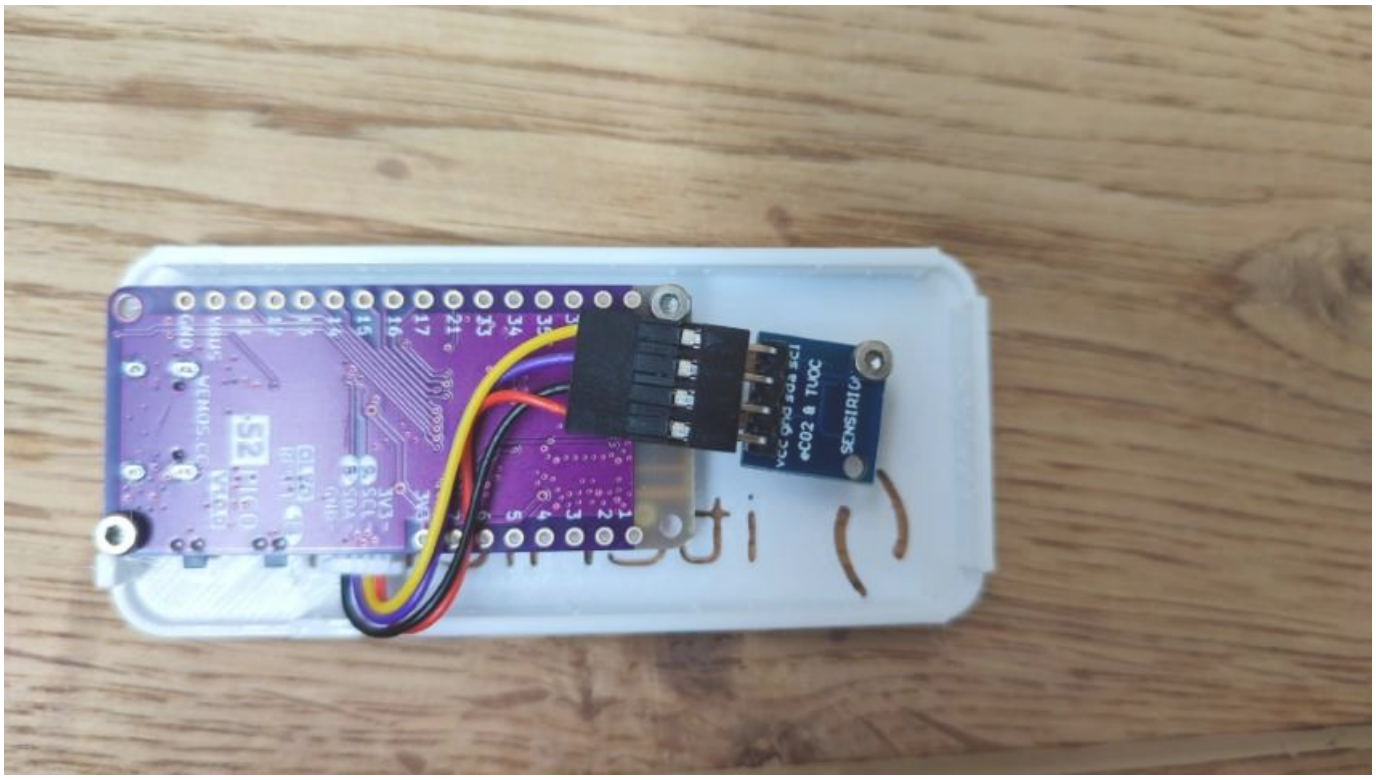


Align the board and spacer with the front of the enclosure and finish screwing them in.



3.6 Mount the sensor in the enclosure

Align the sensor with the enclosure as seen in this picture and screw it in.



3.7 Snap the enclosure to close it

Final step, take the back of the enclosure and snap it closed! Take some care not to force some of the plastic guiding parts.

All done!

itenium 2022