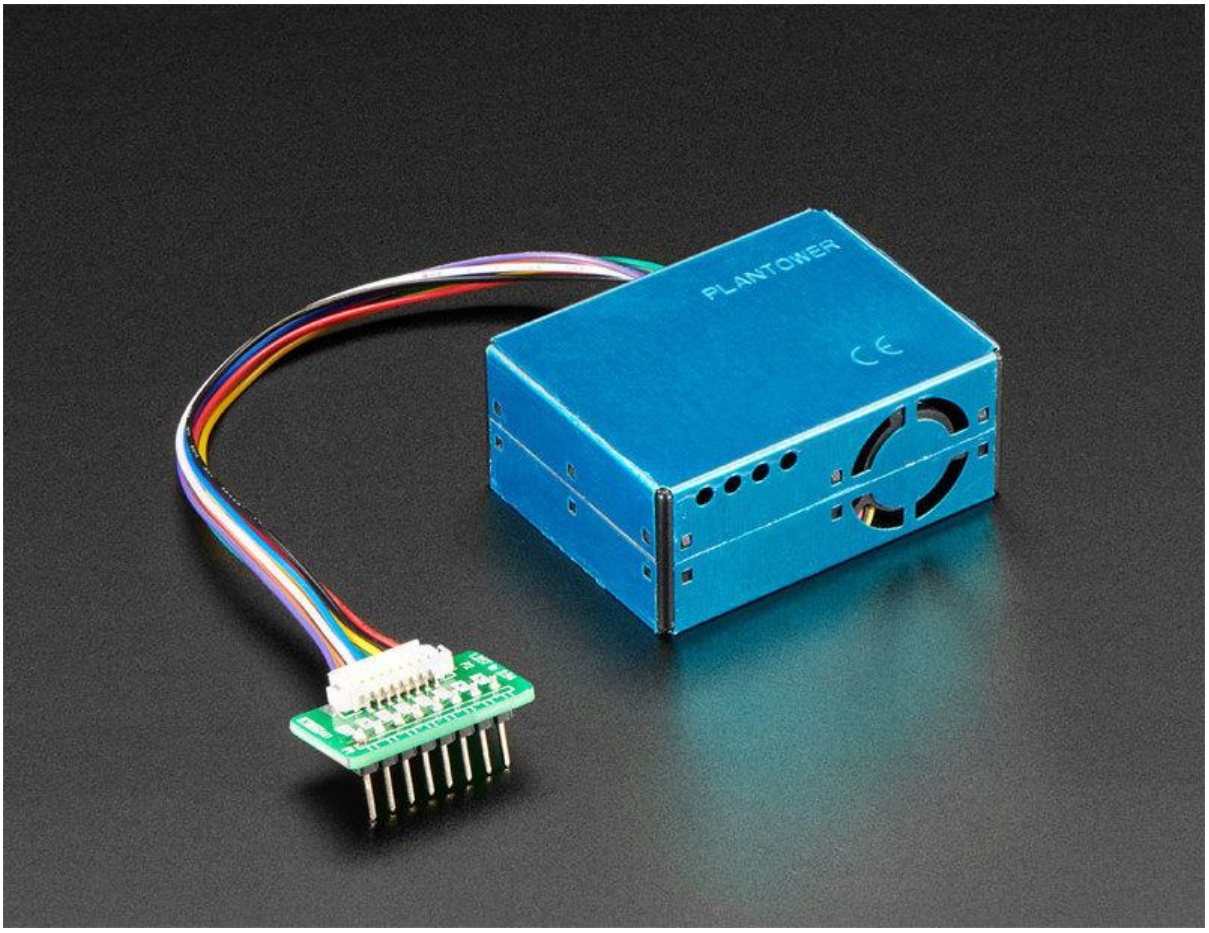
PM2.5 Air Quality Sensor

Created by lady ada



https://learn.adafruit.com/pm25-air-quality-sensor

Last updated on 2021-11-15 07:06:08 PM EST

Table of Contents

Overview 3

Arduino Code 4

- Wiring 5

Python & CircuitPython 8

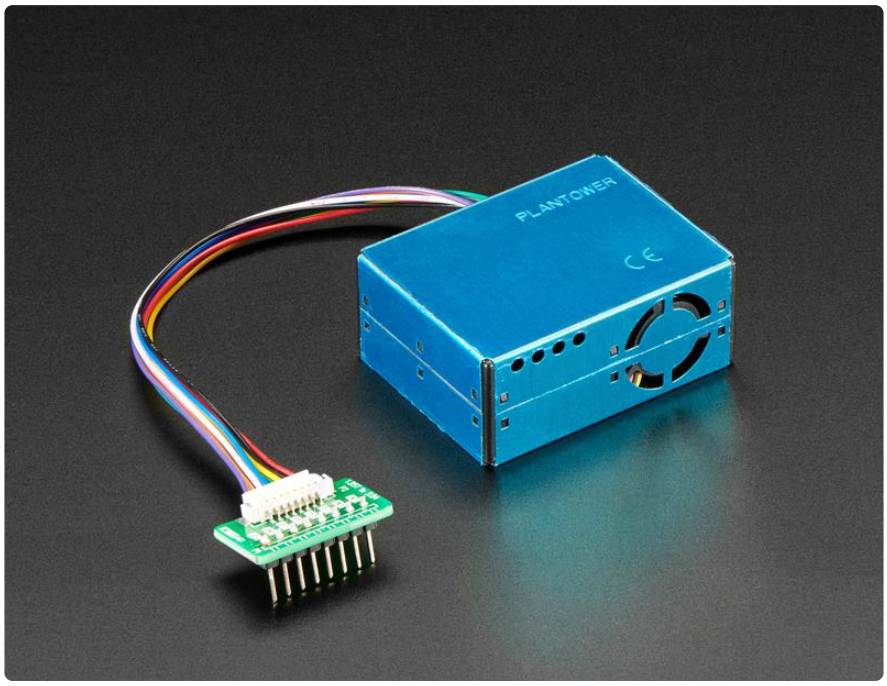
- CircuitPython Microcontroller Wiring 8
- Python Computer Wiring 8
- CircuitPython & Python Usage 10
- CircuitPython Microcontroller 10
- Linux/Computer/Raspberry Pi with Python 10

Usage Notes 12

Downloads 15

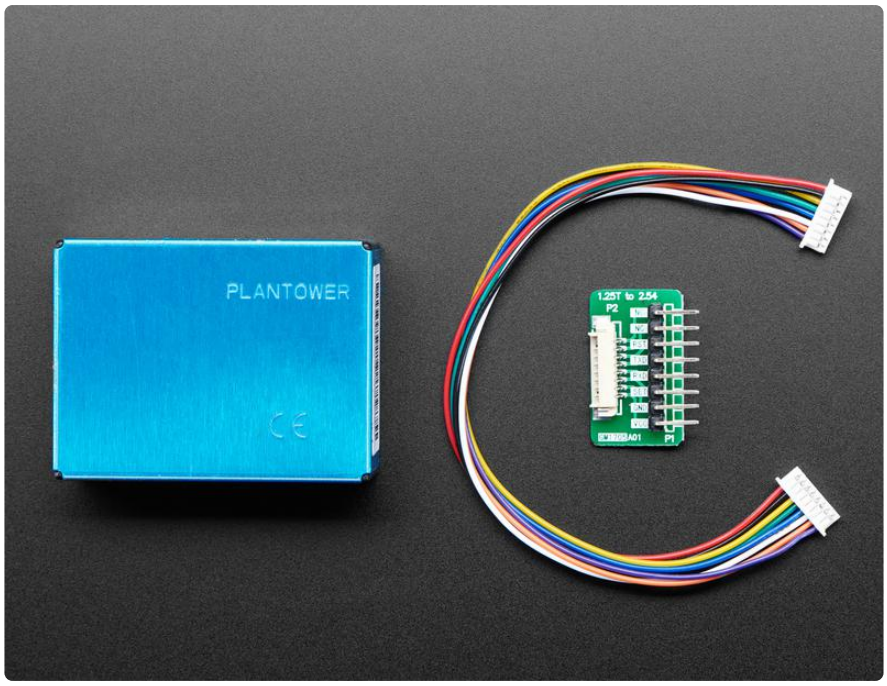
- Files: 15

Overview



Breathe easy, knowing that you can track and sense the quality of the air around you with the PM2.5 Air Quality Sensor with Breadboard Adapter particulate sensor. Mad Max & Furiosa definitely should have hooked up one of these in their truck while scavenging the dusty desert wilderness of post-apocalyptic Australia (https://adafru.it/CiF). And for those of us not living in an Outback dystopia, this sensor + adapter kit is great for monitoring air quality, and super easy to use!

WITNESS real-time, reliable measurement of PM2.5 dust concentrations! (PM2.5 refers to particles that are 2.5 microns or smaller in diameter.) This sensor uses laser scattering to radiate suspending particles in the air, then collects scattering light to obtain the curve of scattering light change with time. The microprocessor calculates equivalent particle diameter and the number of particles with different diameter per unit volume.



You'll need to hook this up to a microcontroller with UART input (or you could theoretically wire it up to a USB-Serial converter and parse the data on a computer (https://adafru.it/C7B)) - we have code for both Arduino and CircuitPython. 9600 baud data streams out once per second, you'll get:

- PM1.0, PM2.5 and PM10.0 concentration in both standard & enviromental units
- Particulate matter per 0.1L air, categorized into 0.3um, 0.5um, 1.0um, 2.5um, 5.0um and 10um size bins

As well as checksum, in binary format (its fairly easy to parse the binary format, but it doesn't come out as pure readable ascii text)

We give you the sensor box as well as the cable and a 0.1" / 2.54mm breakout board so you can wire it easily. You only need power plus one data pin (for the UART TX). Power is 5V, logic is 3.3V

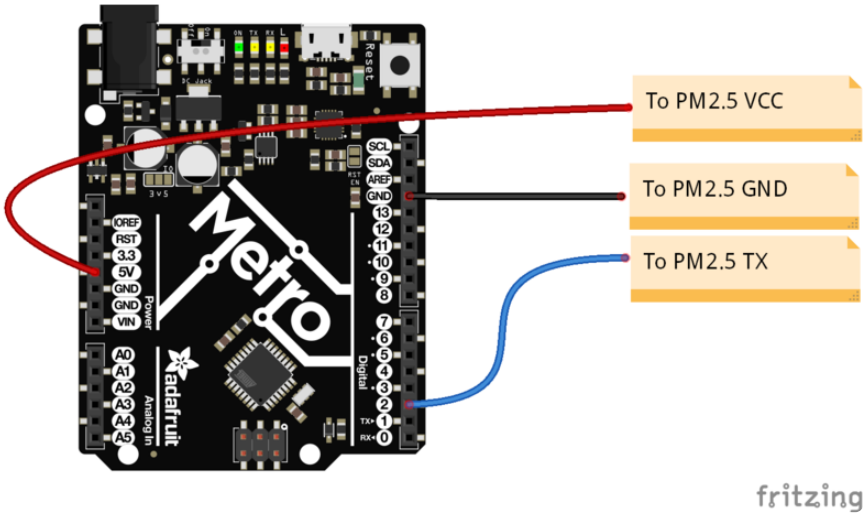
Arduino Code

Using the PM2.5 with Arduino is a simple matter of wiring up it to your Arduino-compatible microcontroller, installing the Adafruit PM25AQI (https://adafru.it/Mej) library we've written, and running the provided example code.

This code will get you started with any Arduino compatible (e.g. Arduino UNO, Adafruit Metro, ESP8266, Teensy, etc. As long as you have either a hardware serial or software serial port that can run at 9600 baud.

Wiring

Wiring is simple! Power the sensor with +5V and GND and then connect the data out pin (3.3V logic) to the serial input pin you'll use. Whether or not you are using hardware or software UART/serial may affect the pin, so adjust that as necessary. This wiring works for ATMega328P-based boards for sure, with Digital #2 as the data pin:



To use this example with the PM2.5 sensor, you'll need to make some changes.

```
/* Test sketch for Adafruit PM2.5 sensor with UART or I2C */

#include "Adafruit_PM25AQI.h"

// If your PM2.5 is UART only, for UNO and others (without hardware serial)
// we must use software serial...
// pin #2 is IN from sensor (TX pin on sensor), leave pin #3 disconnected
// comment these two lines if using hardware serial
//#include <SoftwareSerial.h>
//SoftwareSerial pmSerial(2, 3);

Adafruit_PM25AQI aqi = Adafruit_PM25AQI();

void setup() {
  // Wait for serial monitor to open
  Serial.begin(115200);
  while (!Serial) delay(10);

  Serial.println("Adafruit PMSA003I Air Quality Sensor");

  // Wait one second for sensor to boot up!
  delay(1000);

  // If using serial, initialize it and set baudrate before starting!
  // Uncomment one of the following
  //Serial1.begin(9600);
  //pmSerial.begin(9600);

  // There are 3 options for connectivity!
  if (! aqi.begin_I2C()) {      // connect to the sensor over I2C
  //if (! aqi.begin_UART(&Serial1)) { // connect to the sensor over hardware serial
```

```
  //if (! aqi.begin_UART(&pmSerial)) { // connect to the sensor over software
serial
    Serial.println("Could not find PM 2.5 sensor!");
    while (1) delay(10);
  }

  Serial.println("PM25 found!");
}

void loop() {
  PM25_AQI_Data data;

  if (! aqi.read(&data)) {
    Serial.println("Could not read from AQI");
    delay(500);  // try again in a bit!
    return;
  }
  Serial.println("AQI reading success");

  Serial.println();
  Serial.println(F("---------------------------------------"));
  Serial.println(F("Concentration Units (standard)"));
  Serial.println(F("---------------------------------------"));
  Serial.print(F("PM 1.0: ")); Serial.print(data.pm10_standard);
  Serial.print(F("\t\tPM 2.5: ")); Serial.print(data.pm25_standard);
  Serial.print(F("\t\tPM 10: ")); Serial.println(data.pm100_standard);
  Serial.println(F("Concentration Units (environmental)"));
  Serial.println(F("---------------------------------------"));
  Serial.print(F("PM 1.0: ")); Serial.print(data.pm10_env);
  Serial.print(F("\t\tPM 2.5: ")); Serial.print(data.pm25_env);
  Serial.print(F("\t\tPM 10: ")); Serial.println(data.pm100_env);
  Serial.println(F("---------------------------------------"));
  Serial.print(F("Particles > 0.3um / 0.1L air:"));
Serial.println(data.particles_03um);
  Serial.print(F("Particles > 0.5um / 0.1L air:"));
Serial.println(data.particles_05um);
  Serial.print(F("Particles > 1.0um / 0.1L air:"));
Serial.println(data.particles_10um);
  Serial.print(F("Particles > 2.5um / 0.1L air:"));
Serial.println(data.particles_25um);
  Serial.print(F("Particles > 5.0um / 0.1L air:"));
Serial.println(data.particles_50um);
  Serial.print(F("Particles > 10 um / 0.1L air:"));
Serial.println(data.particles_100um);
  Serial.println(F("---------------------------------------"));


  delay(1000);
}
```

Comment out the following line by adding "`//`" before it:

```
if (! aqi.begin_I2C()) {      // connect to the sensor over I2C
```

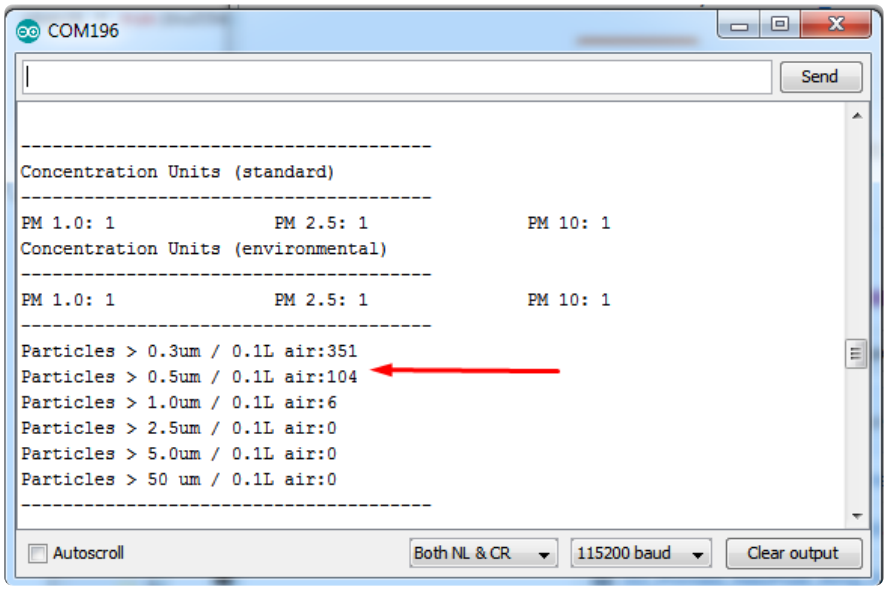
Uncomment the following lines by removing the "`//`" from the beginning:

```
//#include &lt;SoftwareSerial.h&gt;
//SoftwareSerial pmSerial(2, 3);

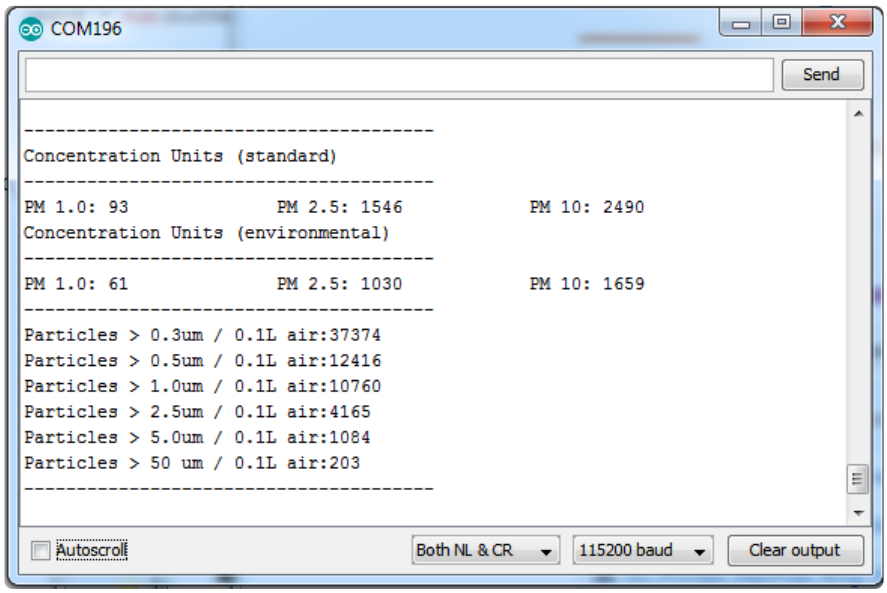
  //pmSerial.begin(9600);

  //if (! aqi.begin_UART(&amp;pmSerial)) { // connect to the sensor over software
serial
```

Once the changes are made, upload this code to your board, and open up the serial console at 115200 baud. You'll see data printed out once a second, with all the measurements. For a clean-air indoor room you'll see something like this:



If you hold up a smoking soldering iron or something else that creates a lot of dust, you'll see much higher numbers!



Note that the numbers are very precise looking but we don't believe that they're going to be perfectly accurate, calibration may be necessary!

Python & CircuitPython

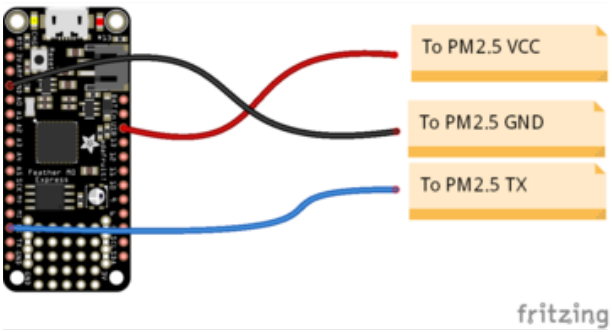
It's easy to use the PM2.5 and the Adafruit CircuitPython PM25 (https://adafru.it/Mek) module. This library allows you to easily write Python code that reads particle concentrations, and particle diameter and the number of particles with different diameters per unit volume.

You can use this sensor with any CircuitPython microcontroller board or with a computer that has GPIO and Python thanks to Adafruit_Blinka, our CircuitPython-for-Python compatibility library (https://adafru.it/BSN).

CircuitPython Microcontroller Wiring

First, connect the sensor to your microcontroller board using UART (a serial port).

Here is an example of it connected to a Feather M0 using UART:



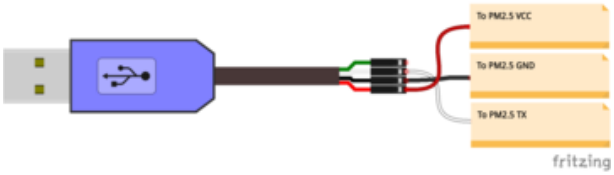
- Sensor VCC to board 5V
- Sensor GND to board GND
- Sensor TX to board RX

Remember: RX does not connect to RX!

Python Computer Wiring

Since there's dozens of Linux computers/boards you can use we will show wiring for Raspberry Pi. For other platforms, please visit the guide for CircuitPython on Linux to see whether your platform is supported (https://adafru.it/BSN).

Here you have two options: An external USB-to-serial converter, or the built-in UART on the Pi's RX pin. Here's an example of wiring up the USB-to-serial converter (https://adafru.it/dDd):

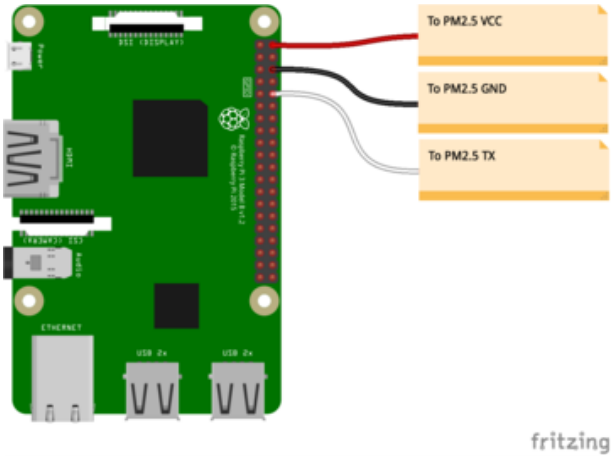


- Sensor VCC to USB 5V
- Sensor GND to USB GND
- Sensor TX to USB RX (white wire)

Remember: RX does not connect to RX!

For single board computers other than the Raspberry Pi, the serial port may be tied to the console or not be available to the user. Please see the board documentation to see how the serial port may be used

Here's an example using the Pi's built-in UART:



- Sensor VCC to Pi 5V
- Sensor GND to Pi GND
- Sensor TX to Pi RX

Remember: RX does not connect to RX!

If you want to use the built-in UART, you'll need to disable the serial console and enable the serial port hardware in raspi-config. See the UART/Serial section of the CircuitPython on Raspberry Pi guide (https://adafru.it/CEk) for detailed instructions on how to do this.

To use the pm25_simpletest.py with the PM2.5 sensor, you'll have to make some changes.

CircuitPython & Python Usage

To demonstrate the PM2.5 in CircuitPython and Python, let's look at a complete program example.

CircuitPython Microcontroller

With a CircuitPython microcontroller, save this file as code.py on your board. Then comment out the following lines by inserting a '#' before each one:

```
i2c = busio.I2C(board.SCL, board.SDA, frequency=100000)

pm25 = adafruit_pm25.i2c.PM25_I2C(i2c, reset_pin)
```

And uncomment the following lines by removing the '# ' (hash and space both!) before each one:

```
# uart = busio.UART(board.TX, board.RX, baudrate=9600)

# pm25 = adafruit_pm25.uart.PM25_UART(uart, reset_pin)
```

Then, open up the serial console (https://adafru.it/Bec) to see its output.

Linux/Computer/Raspberry Pi with Python

When using a USB to serial cable or a Raspberry Pi, comment out the following lines by inserting a '#' before each one:

```
i2c = busio.I2C(board.SCL, board.SDA, frequency=100000)

pm25 = adafruit_pm25.i2c.PM25_I2C(i2c, reset_pin)
```

For Raspberry Pi, uncomment the following lines by removing the '# ' (hash and space both!) before each one:

```
# import serial
# uart = serial.Serial("/dev/ttyS0", baudrate=9600, timeout=0.25)
```

For a USB to serial cable, uncomment the following lines by removing the '# ' (hash and space both!) before each one:

```
# import serial
# uart = serial.Serial("/dev/ttyUSB0", baudrate=9600, timeout=0.25)
```

Install the python `serial` with library with

`pip3 install pyserial`

Now you're ready to run the program with the following command:

```
python3 pm25_simpletest.py
```

```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

"""
Example sketch to connect to PM2.5 sensor with either I2C or UART.
"""

# pylint: disable=unused-import
import time
import board
import busio
from digitalio import DigitalInOut, Direction, Pull
from adafruit_pm25.i2c import PM25_I2C


reset_pin = None
# If you have a GPIO, its not a bad idea to connect it to the RESET pin
# reset_pin = DigitalInOut(board.G0)
# reset_pin.direction = Direction.OUTPUT
# reset_pin.value = False


# For use with a computer running Windows:
# import serial
# uart = serial.Serial("COM30", baudrate=9600, timeout=1)

# For use with microcontroller board:
# (Connect the sensor TX pin to the board/computer RX pin)
# uart = busio.UART(board.TX, board.RX, baudrate=9600)

# For use with Raspberry Pi/Linux:
# import serial
# uart = serial.Serial("/dev/ttyS0", baudrate=9600, timeout=0.25)

# For use with USB-to-serial cable:
# import serial
# uart = serial.Serial("/dev/ttyUSB0", baudrate=9600, timeout=0.25)

# Connect to a PM2.5 sensor over UART
# from adafruit_pm25.uart import PM25_UART
# pm25 = PM25_UART(uart, reset_pin)

# Create library object, use 'slow' 100KHz frequency!
i2c = busio.I2C(board.SCL, board.SDA, frequency=100000)
# Connect to a PM2.5 sensor over I2C
pm25 = PM25_I2C(i2c, reset_pin)

print("Found PM2.5 sensor, reading data...")

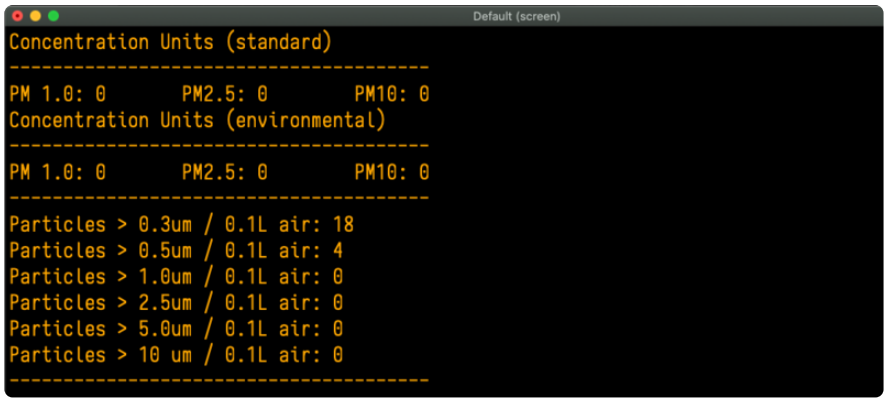
while True:
```

```
    time.sleep(1)

    try:
        aqdata = pm25.read()
        # print(aqdata)
    except RuntimeError:
        print("Unable to read from sensor, retrying...")
        continue

    print()
    print("Concentration Units (standard)")
    print("---------------------------------------")
    print(
        "PM 1.0: %d\tPM2.5: %d\tPM10: %d"
        % (aqdata["pm10 standard"], aqdata["pm25 standard"], aqdata["pm100
standard"])
    )
    print("Concentration Units (environmental)")
    print("---------------------------------------")
    print(
        "PM 1.0: %d\tPM2.5: %d\tPM10: %d"
        % (aqdata["pm10 env"], aqdata["pm25 env"], aqdata["pm100 env"])
    )
    print("---------------------------------------")
    print("Particles > 0.3um / 0.1L air:", aqdata["particles 03um"])
    print("Particles > 0.5um / 0.1L air:", aqdata["particles 05um"])
    print("Particles > 1.0um / 0.1L air:", aqdata["particles 10um"])
    print("Particles > 2.5um / 0.1L air:", aqdata["particles 25um"])
    print("Particles > 5.0um / 0.1L air:", aqdata["particles 50um"])
    print("Particles > 10 um / 0.1L air:", aqdata["particles 100um"])
    print("---------------------------------------")
```

You should see output looking something like the following:



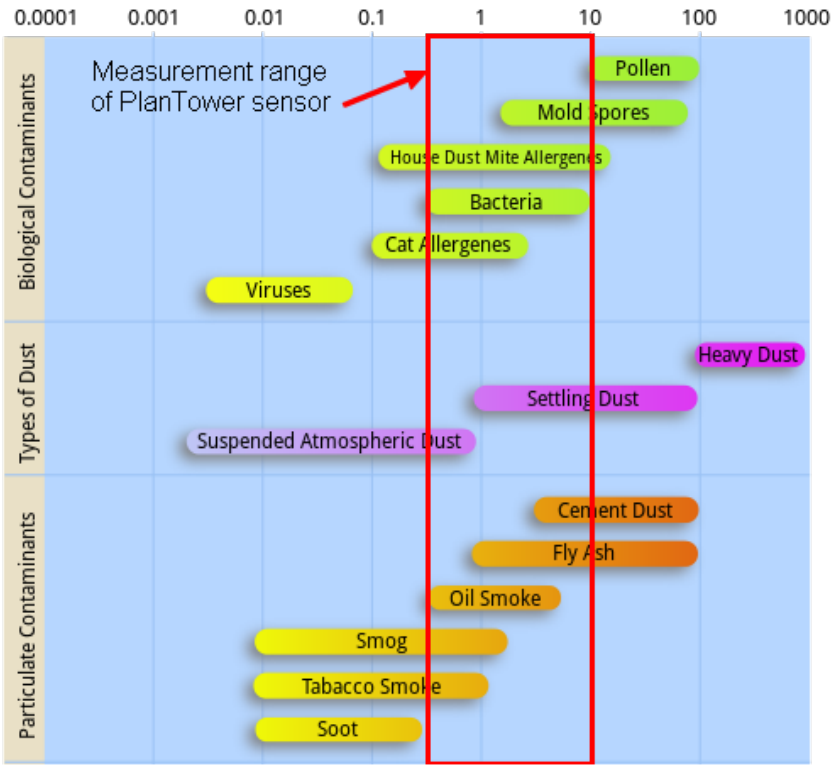
That's all there is to using the PM2.5 air quality sensor with CircuitPython!

Usage Notes

StanJ wrote up an amazing analysis report of using the PM2.5 sensor in their lab (https://adafru.it/FDX), and we think its helpful for others to understand what and how the sensor works and what to expect from it! We've duplicated it here as well:

I've read quite a lot on the PlanTower sensors, although I'm nothing like an expert :-). The CF readings are 'Calibration Factory' and aren't useful; the 'Environmental' or 'Ambient' concentration readings are the data you want for air quality measurements. I'm using the PMS5003 for a continuous check on cleanroom quality, so I only use the raw Particle Counts as that's the measurement specified in ISO 14644-1 .

As Solaria123 noted in viewtopic.php?f=19&t=135496 (https://adafru.it/doW), the sensor estimates particles > 2.5um and doesn't (or can't) measure them. The article at ResearchGate showed that a concentration composed solely of larger particles wasn't seen by the sensor. For our cleanroom use that's OK as the HEPA filters are more efficient as the particle size increases. For non-filtered air it's a bit more of concern as the different particle sizes are composed of different pollutants, so you might be missing a pollutant if it's composed primarily of larger particles like pollen.



One amusing note in the translated PlanTower datasheet is "Only the consistency among the PM sensors of PLANTOWER is promised and ensured. And the sensor should not be checked with any third party equipment." Several groups including AQICN.org have done exactly that, and we have as well. The PlanTower sensor compares favorably with the readings from our calibrated Beckman Particle Counter, although the 30-50% uncertainty on the PlanTower 0.3 and 0.5 um bins means you can't get an exact comparison. We're only using the sensor for a rough check on

current air quality, not to verify compliance with ISO 14644.

A frustrating artifact of the PlanTower sensor is the sampling rate versus data output. With small change between readings the sensor only updates the counts every 2.3 seconds, although it outputs data every second. That means it may duplicate over half of the data, with no way to verify whether any reading is a duplicate. For a normal home or outdoor setting you could simply discard any reading when the checksum is identical to the previous data, as you're highly unlikely to have two successive samples with the same values. In a cleanroom we're looking at very low particle counts, and two successive samples might well be identical. The only way I could get around that is by throwing away 2 of every 3 data packets to insure I'm getting real counts, which increases the total sample time. I add the results from 100 unique 0.1 liter samples to get a reading of particles in 10 liters of air for my measurement, which means 300 samples with 2/3rds of the data thrown away.

```
amb=[003a 005c 0061] raw=[386a 1160 0325 004c 000b 0001] csum=0542
amb=[003b 005d 0063] raw=[38cd 1175 033c 0054 000e 0004] csum=05ea
amb=[003c 0060 0065] raw=[398a 11ba 033c 0054 000a 0003] csum=05f4
amb=[003c 0060 0066] raw=[3a8c 120f 0340 0050 000d 0003] csum=0555
amb=[003d 0060 0066] raw=[3b04 122e 0333 0050 000d 0003] csum=04e1
amb=[003c 005e 0064] raw=[3b04 122a 0339 0056 000b 0003] csum=04dc
amb=[003c 005e 0064] raw=[3b04 122a 0339 0056 000b 0003] csum=04dc duplicate
amb=[003c 005e 0064] raw=[3b04 122a 0339 0056 000b 0003] csum=04dc duplicate
amb=[003c 005c 0062] raw=[3b22 1232 0330 004b 000a 0003] csum=04e2
amb=[003c 005c 0062] raw=[3b22 1232 0330 004b 000a 0003] csum=04e2 duplicate
amb=[003c 005c 0062] raw=[3b22 1232 0330 004b 000a 0003] csum=04e2 duplicate
amb=[003b 0059 005f] raw=[3a7a 1211 030e 0043 000a 0003] csum=04de
amb=[003a 0058 005e] raw=[3a7a 1211 030e 0043 000a 0003] csum=04d8
amb=[003a 0058 005e] raw=[3a7a 1211 030e 0043 000a 0003] csum=04d8 duplicate
amb=[003a 0058 005e] raw=[3a35 11fa 030c 003b 0009 0003] csum=056e
```

What you're seeing above is the 1 second data window sliding along the (typical) 2.3 second sampling window. When the data changes significantly between samples the sensor shortens the sample window to 200-800ms, which may be why the first 6 data points show unique numbers (faster sampling rate).

The readings above are in my home, and I smoke so the particle counts vary wildly about 1000:1 over time with a decent quality air filter. When I'm home I run the air handler fan continuously to level out the temperature over the house, and when I'm away I let the fan cycle with the AC or heat. You can see the difference below in how rapidly the particle counts fall off with continuous filtering. The rapid fall off continuous curve is [sleeping], and the slow fall off is cycling [away from home]. Data points are every 30 minutes.



Downloads

Files:

- PMS5003 Datasheet / Manual (https://adafru.it/CiH)