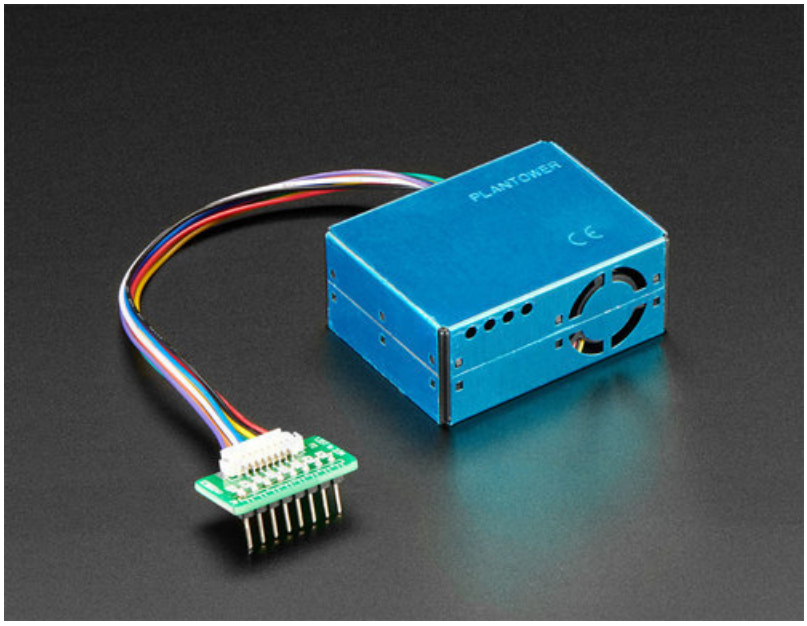
adafruit learning system

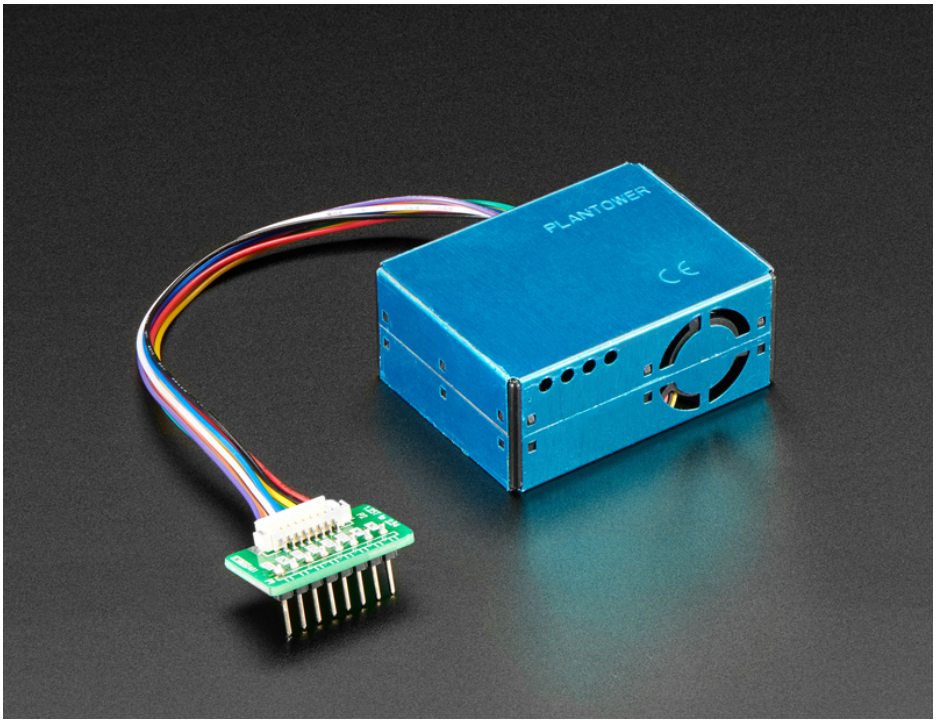
PM2.5 Air Quality Sensor

Created by lady ada



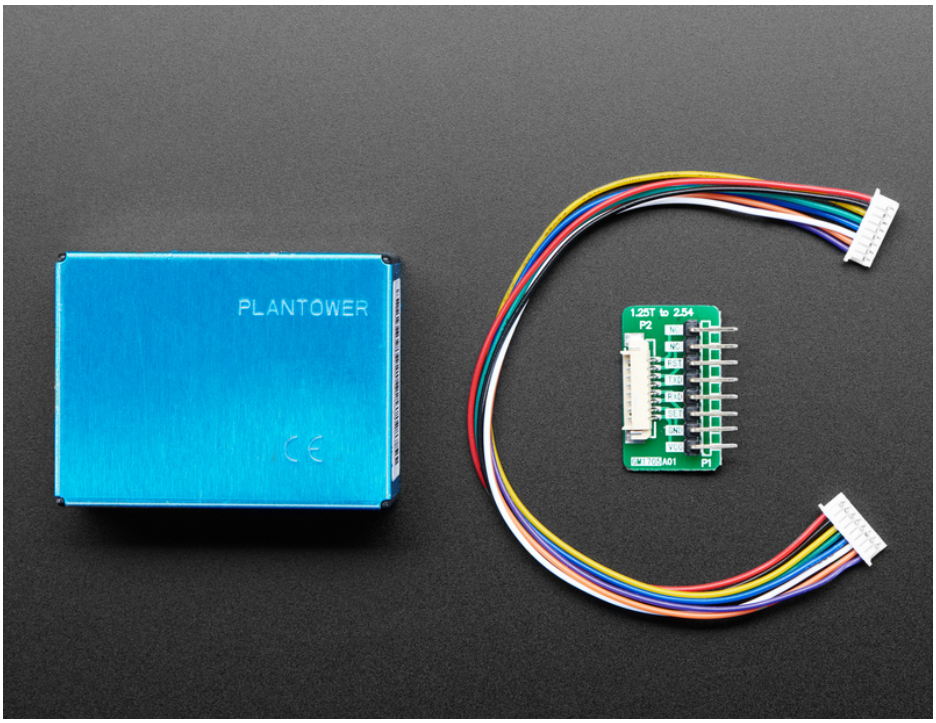
Last updated on 2019-08-25 05:59:13 PM UTC

Overview



Breathe easy, knowing that you can track and sense the quality of the air around you with th**e 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)

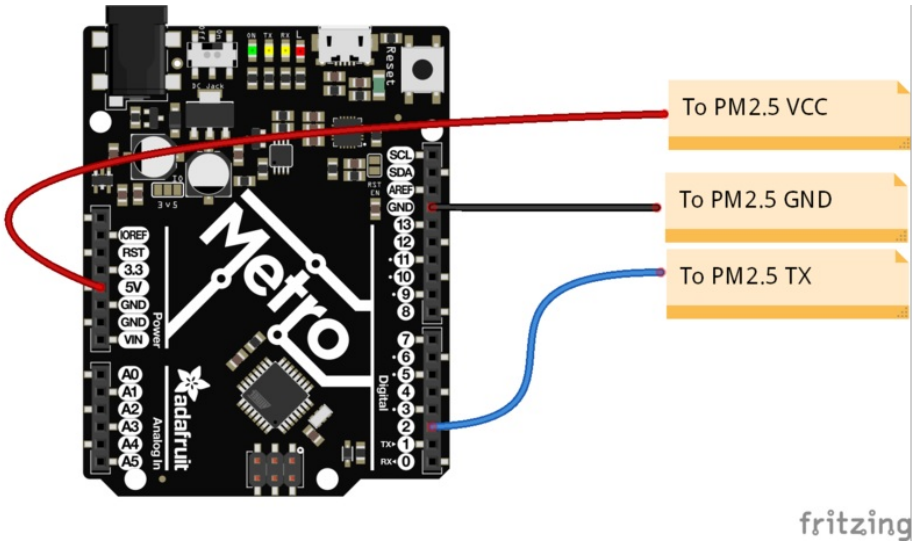
We give you the sensor box as well as the cable and a 0.1" / 2.54mm breakout board so you can wire it easily. You only need power plus one data pin (for the UART TX). Power is 5V, logic is 3.3V

Arduino Code

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:



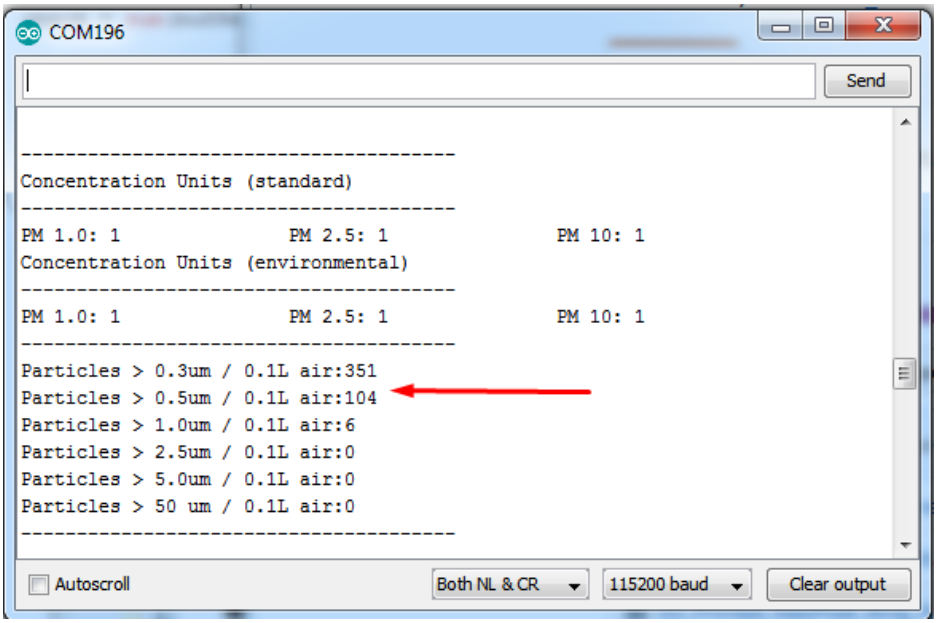
Upload this code to your board, and open up the serial console at **115200** baud

```
// On Leonardo/Micro or others with hardware serial, use those!
// uncomment this line:
// #define pmsSerial Serial1

// 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 pmsSerial(2, 3);

void setup() {
  // our debugging output
  Serial.begin(115200);

  // sensor baud rate is 9600
  pmsSerial.begin(9600);
}

struct pms5003data {
  uint16_t framelen;
  uint16_t pm10_standard, pm25_standard, pm100_standard;
  uint16_t pm10_env, pm25_env, pm100_env;
  uint16_t particles_03um, particles_05um, particles_10um, particles_25um, particles_50um, particles_100u
  uint16_t unused;
```

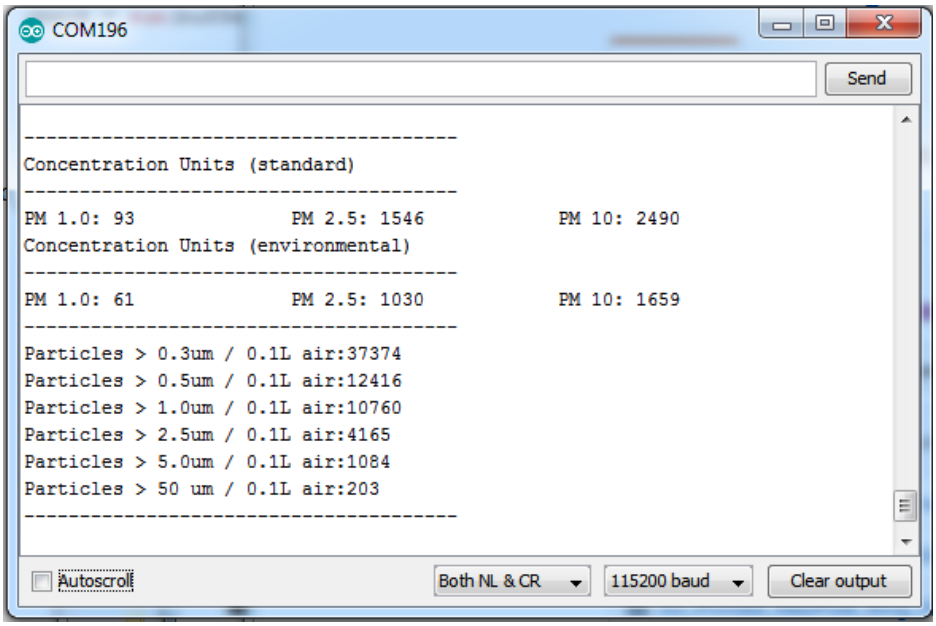
```
  uint16_t checksum;
};

struct pms5003data data;

void loop() {
  if (readPMSdata(&pmsSerial)) {
    // reading data was successful!
    Serial.println();
    Serial.println("---------------------------------------");
    Serial.println("Concentration Units (standard)");
    Serial.print("PM 1.0: "); Serial.print(data.pm10_standard);
    Serial.print("\t\tPM 2.5: "); Serial.print(data.pm25_standard);
    Serial.print("\t\tPM 10: "); Serial.println(data.pm100_standard);
    Serial.println("---------------------------------------");
    Serial.println("Concentration Units (environmental)");
    Serial.print("PM 1.0: "); Serial.print(data.pm10_env);
    Serial.print("\t\tPM 2.5: "); Serial.print(data.pm25_env);
    Serial.print("\t\tPM 10: "); Serial.println(data.pm100_env);
    Serial.println("---------------------------------------");
    Serial.print("Particles > 0.3um / 0.1L air:"); Serial.println(data.particles_03um);
    Serial.print("Particles > 0.5um / 0.1L air:"); Serial.println(data.particles_05um);
    Serial.print("Particles > 1.0um / 0.1L air:"); Serial.println(data.particles_10um);
    Serial.print("Particles > 2.5um / 0.1L air:"); Serial.println(data.particles_25um);
    Serial.print("Particles > 5.0um / 0.1L air:"); Serial.println(data.particles_50um);
    Serial.print("Particles > 10.0 um / 0.1L air:"); Serial.println(data.particles_100um);
    Serial.println("---------------------------------------");
  }
}

boolean readPMSdata(Stream *s) {
  if (! s->available()) {
    return false;
  }

  // Read a byte at a time until we get to the special '0x42' start-byte
  if (s->peek() != 0x42) {
    s->read();
    return false;
  }

  // Now read all 32 bytes
  if (s->available() < 32) {
    return false;
  }

  uint8_t buffer[32];
  uint16_t sum = 0;
  s->readBytes(buffer, 32);

  // get checksum ready
  for (uint8_t i=0; i<30; i++) {
    sum += buffer[i];
  }

  /* debugging
  for (uint8_t i=2; i<32; i++) {
    Serial.print("0x"); Serial.print(buffer[i], HEX); Serial.print(", ");
  }
  Serial.println();
```

```
  */

  // The data comes in endian'd, this solves it so it works on all platforms
  uint16_t buffer_u16[15];
  for (uint8_t i=0; i<15; i++) {
    buffer_u16[i] = buffer[2 + i*2 + 1];
    buffer_u16[i] += (buffer[2 + i*2] << 8);
  }

  // put it into a nice struct :)
  memcpy((void *)&data, (void *)buffer_u16, 30);

  if (sum != data.checksum) {
    Serial.println("Checksum failure");
    return false;
  }
  // success!
  return true;
}
```

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!

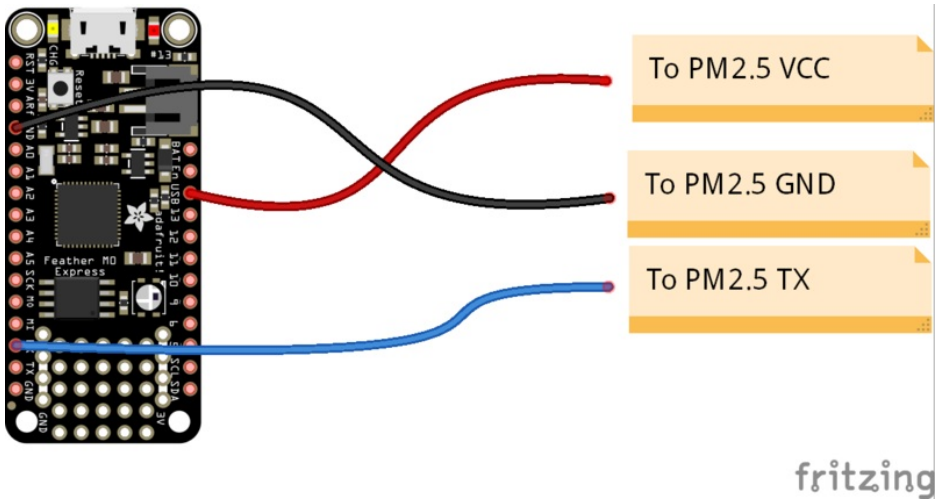


Note that the numbers are very precise looking but we don't believe that they're going to be perfectly accurate, calibration may be necessary!

CircuitPython Code

This code's pretty simple, you can hook up the power and ground pins to 5V and GND, respectively

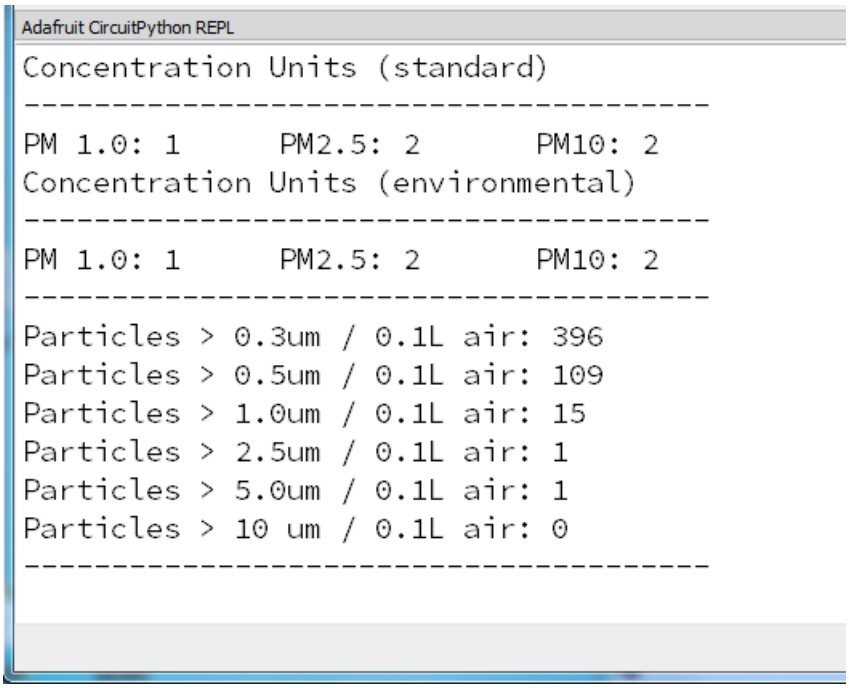
Then wire up the **TX** pin from the sensor to **RX** pin on your CircuitPython board. Note, **RX** does not connect to **RX**!




```
import board
import busio
from digitalio import DigitalInOut, Direction

try:
    import struct
except ImportError:
    import ustruct as struct

led = DigitalInOut(board.D13)
led.direction = Direction.OUTPUT

# Connect the Sensor's TX pin to the board's RX pin
uart = busio.UART(board.TX, board.RX, baudrate=9600)

buffer = []

while True:
    data = uart.read(32)  # read up to 32 bytes
    data = list(data)
    # print("read: ", data)          # this is a bytearray type

    buffer += data

    while buffer and buffer[0] != 0x42:
        buffer.pop(0)

    if len(buffer) > 200:
        buffer = []  # avoid an overrun if all bad data
    if len(buffer) < 32:
        continue

    if buffer[1] != 0x4d:
        buffer.pop(0)
```

```
        continue

    frame_len = struct.unpack(">H", bytes(buffer[2:4]))[0]
    if frame_len != 28:
        buffer = []
        continue

    frame = struct.unpack(">HHHHHHHHHHHHHH", bytes(buffer[4:]))

    pm10_standard, pm25_standard, pm100_standard, pm10_env, \
        pm25_env, pm100_env, particles_03um, particles_05um, particles_10um, \
        particles_25um, particles_50um, particles_100um, skip, checksum = frame

    check = sum(buffer[0:30])

    if check != checksum:
        buffer = []
        continue

    print("Concentration Units (standard)")
    print("---------------------------------------")
    print("PM 1.0: %d\tPM2.5: %d\tPM10: %d" %
          (pm10_standard, pm25_standard, pm100_standard))
    print("Concentration Units (environmental)")
    print("---------------------------------------")
    print("PM 1.0: %d\tPM2.5: %d\tPM10: %d" % (pm10_env, pm25_env, pm100_env))
    print("---------------------------------------")
    print("Particles > 0.3um / 0.1L air:", particles_03um)
    print("Particles > 0.5um / 0.1L air:", particles_05um)
    print("Particles > 1.0um / 0.1L air:", particles_10um)
    print("Particles > 2.5um / 0.1L air:", particles_25um)
    print("Particles > 5.0um / 0.1L air:", particles_50um)
    print("Particles > 10 um / 0.1L air:", particles_100um)
    print("---------------------------------------")

    buffer = buffer[32:]
    # print("Buffer ", buffer)
```

Then open up the REPL (https://adafru.it/Bec) to see the data printed out nicely for you!


Adafruit CircuitPython REPL
Concentration Units (standard)

PM 1.0: 1 PM2.5: 2 PM10: 2
Concentration Units (environmental)

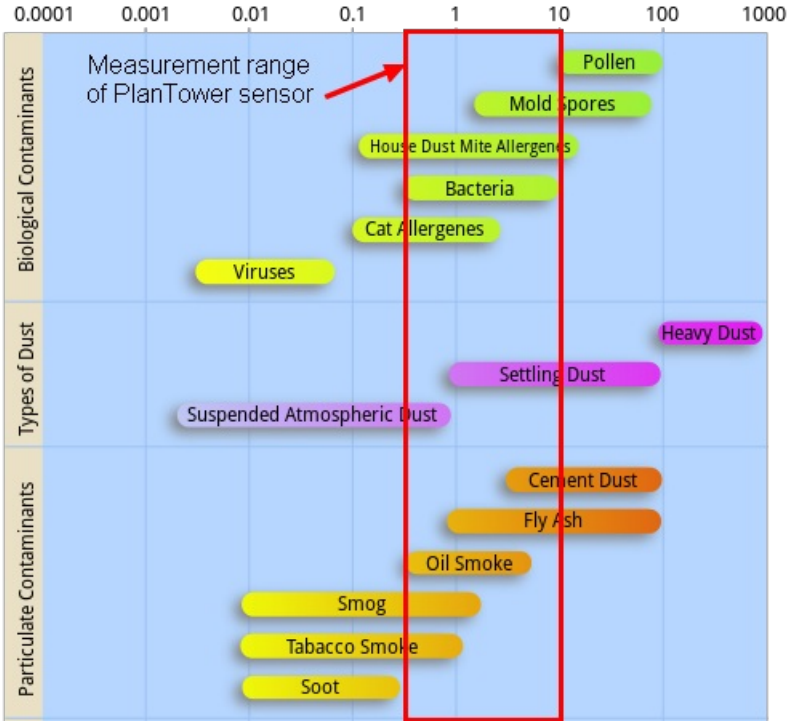
PM 1.0: 1 PM2.5: 2 PM10: 2

Particles > 0.3um / 0.1L air: 396
Particles > 0.5um / 0.1L air: 109
Particles > 1.0um / 0.1L air: 15
Particles > 2.5um / 0.1L air: 1
Particles > 5.0um / 0.1L air: 1
Particles > 10 um / 0.1L air: 0

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)