## Microcontrollers and Sensors: PMS5003 Air Quality Sensor

### Learning outcomes

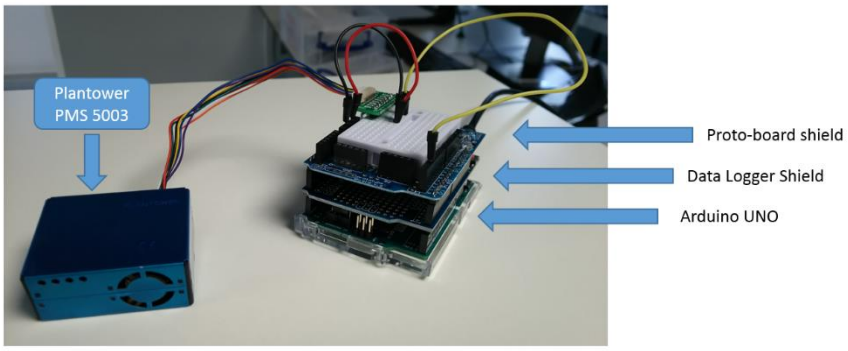
1. Build a complete, self-contained, portable air-quality monitoring station based on the Arduino UNO, PMS5003 Sensor, Data Logger shield and proto-shield
2. Use the Arduino to capture data from the PMS5003 and write this to a file on an SD Card.

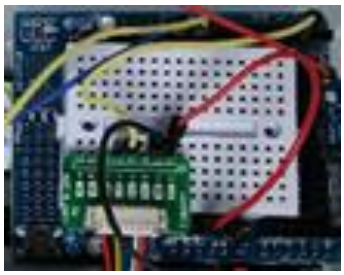
### Equipment

1 x Arduino Uno & USB cable
1 x Data Logger Shield
1 x SD Card
1 x proto-shield
1 x PMS5003 Sensor, cable and connector
1 x USB Power Bank
Velcro
Wires
1 x Plastic Box
2 x elastic bands
1 x length of string
PC with Arduino Integrated Development Environment (IDE)

First you are going to assemble the Arduino "stack".

**Version 1. For PMS Connector with right-angled pins.**
*Carefully* mount the Data Logger shield onto the Arduino UNO. Then mount the proto-board shield on top of that, as below.



The connector can be placed directly into the proto-board as above and left. The three connections to the Arduino/shield are:

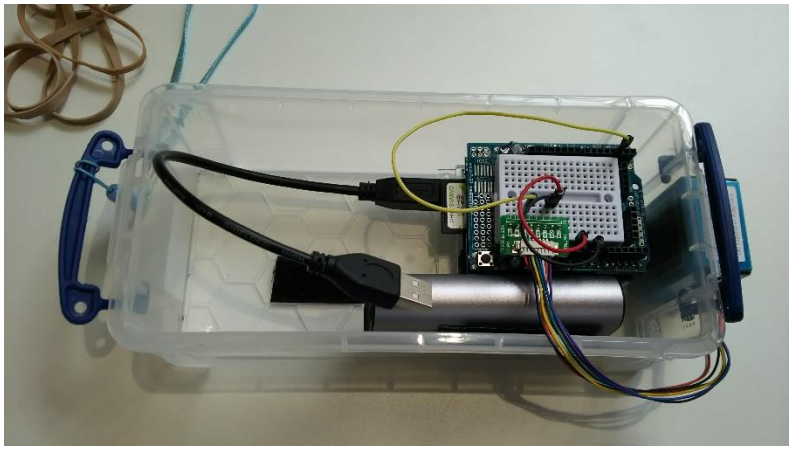
VCC – 5 V
GND – GND
TXD – PIN 2

**Version 2. For PMS Connector with straight pins.**
*Carefully* mount the Data Logger shield onto the Arduino UNO. Then use three short male-female adapter wires to connect between the pins of the PMS connector and the Arduino/shield. The Connections are:

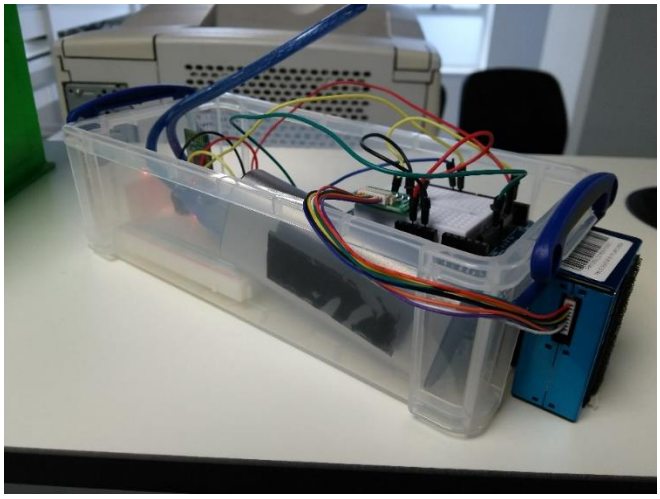
VCC – 5 V
GND – GND
TXD – PIN 2

Next:

Cut 3cm strips of Velcro (3 cm hook type & 3 cm loop type) and attach these to the bottom of the sensor and the Arduino stack (hook type) and the appropriate places on the plastic box (loop type). Do the same with the USB Power Bank – taking care to ensure that you have everything the right way around (illustration shows the version 1 stack).



The sensor goes on the outside of the box near the Arduino stack (illustration shows the version 1 stack).

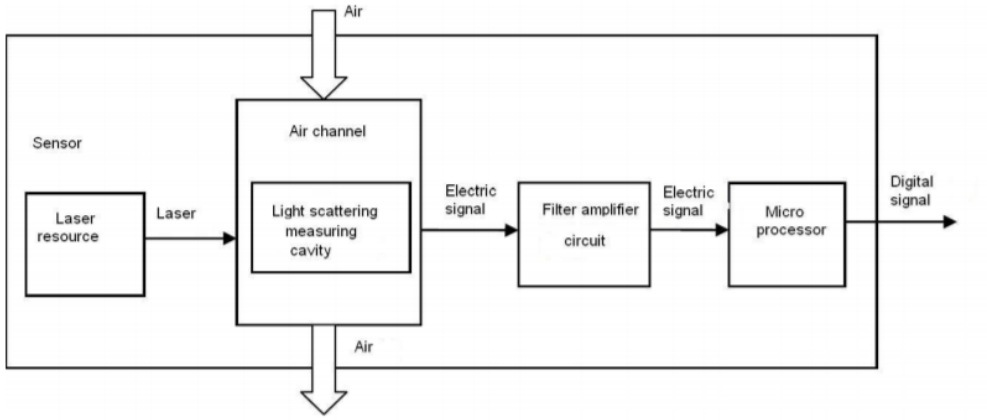


## Testing the Build

Place an SD card into the Data logger and run the CardInfo sketch from the Arduino IDE examples – **remember to change the chipselect pin from 4 to 10 for this data logger**. Confirm that the Arduino detects the SD card.

## Acquiring Data from the PMS5003

The PMS5003 is a much-more sophisticated device than the TEMT6000 light sensor you have used previously. As the block diagram below shows, it contains its own on-board CPU.




This allows it to capture a total of 12 measurements in each cycle, listed in the table below.

| Sensor measurements made by the Plantower PMS5003 | | | |
|---|---|---|---|
| 1 | PM1.0 µg/m$^3$ under standard conditions | 7 | Number of particles < 0.3 µm/l |
| 2 | PM2.5 µg/m$^3$ under standard conditions | 8 | Number of particles < 0.5 µm/l |
| 3 | PM10 µg/m$^3$ under standard conditions | 9 | Number of particles < 1.0 µm/l |
| 4 | PM1.0 µg/m$^3$ under environmental conditions | 10 | Number of particles < 2.5 µm/l |
| 5 | PM2.5 µg/m$^3$ under environmental conditions | 11 | Number of particles < 5.0 µm/l |
| 6 | PM10 µg/m$^3$ under environmental conditions | 12 | Number of particles < 10.0 µm/l |

The CPU bundles these reading together to create a **data packet** that can be exported through the TX/TXD pin of the sensor (actually, strictly speaking it's a "**data frame"** not a data packet, as this is serial communication, but it's easy to visualise it as a packet). The packet is 32 bytes in length in total. The complete structure of the packet can be found in the appendix of the PMS5003 manual included on the KEATS module page; it is enough to know here that each of the 12 sensor

measurements recorded above is stored in a 16-bit-long section of the data, and there are 6 additional bytes for frame-length checksum etc.

In order for the Arduino to read the data from the PMS5003, it must be told where to look for each sensor measurement in the 32-byte data packet. The sketch for reading data from the PMS5003 therefore contains an element called a **structure** ("struct") that details each element in terms of an *unsigned integer* 16 bits long – **uint16_t**; all 32 bytes are described in the structure. The relevant code fragment can be found below:

```
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_100um;
  uint16_t unused;
  uint16_t checksum;
};
```

This structure can be easily understood by comparing it to the description of the data packet found in the appendix to the PMS5003 manual.

Key to the Arduino correctly unpacking the data packet is it receiving the whole packet each cycle. To confirm this, a checksum operation is carried out after each read operation in the sketch – the Arduino looks for the checksum data, which is the data contained in the last 16-bits of each data package, as can be seen from the *struct* above. Similarly, the Arduino must know where the data packet starts, so it looks for a "0x42" byte that marks the start of the packet.

**Fortunately for us, someone has already gone to the trouble of marrying the PMS5003 to an Arduino and produced a sketch for reading from it. That sketch is the "PMS_Basic" sketch on the KEATS module page, and is included in these notes as an appendix.**

Assemble the monitoring station and then copy-paste the PMS_Basic code to the Arduino IDE and upload this sketch to the station. **Open the Serial Monitor and change the BAUD rate from 9600 to 115200** – this is the data rate for the decode of the PMS5003 data packet in the sketch.

If everything is working correctly, you should see the PM2.5 & PM10 particle levels displayed on the monitor. **Ask the TA to help you test the sensor with the smoke generator.**

Having confirmed both that the Arduino detects the SD card and captures data from the PMS5003, you can now upload the **PMS_withSD** sketch, which will allow you to write your PM2.5 & PM10 data to a CSV file on the card. The full sketch can be found on the KEATS module page, and is included in these notes in the appendix.

**Finally, you can switch to powering the system with the USB power bank, giving you a complete, self-contained portable device.**

# Appendix: Arduino sketches

## 1. Reading from the PMS5003 & Displaying PM2.5 & PM10 on Serial Monitor

```
// Basic code for reading data from PMS5003 sensor and displaying on serial monitor

// adapted from https://learn.adafruit.com/pm25-air-quality-sensor/arduino-code
// pin #2 is IN from sensor (TX pin on sensor), leave pin #3 disconnected


#include <SoftwareSerial.h>
SoftwareSerial pmsSerial(2, 3);

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

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

// this tells the Arduino the structure of the data from the sensor
// we could print out any of these values, but all we want are the env pm25 and pm100 values

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_100um;
  uint16_t unused;
  uint16_t checksum;
};

struct pms5003data data;

void loop() {
  if (readPMSdata(&pmsSerial)) {
    // reading data was successful!
    Serial.print("\t\tPM 2.5: "); Serial.println(data.pm25_env);
   Serial.print("\t\tPM 10: "); Serial.println(data.pm100_env);
  }
}
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;
}
```

## 2. Reading from the PMS5003 & writing PM2.5 & PM10 to a CSV file on the SD Card

```
// Basic code for reading data from PMS5003 sensor and displaying on serial monitor and writing to SD Card

// adapted from https://learn.adafruit.com/pm25-air-quality-sensor/arduino-code
// and https://maker.pro/arduino/tutorial/how-to-make-an-arduino-sd-card-data-logger-for-temperature-sensor-data
// and http://www.toptechboy.com/arduino/arduino-lesson-21-log-sensor-data-to-an-sd-card/
// and https://learn.adafruit.com/adafruit-data-logger-shield/using-the-real-time-clock-3
//
// Jamie Barras June 2019

// pin #2 is IN from sensor (TX pin on sensor), leave pin #3 disconnected


#include <SoftwareSerial.h>
#include <SD.h> // the two libraries we need for communicating and writing to the SD card
#include <SPI.h>

File sdcard_file;
int CS_pin = 10; // Pin 10 on Arduino Uno for data logger shield

SoftwareSerial pmsSerial(2, 3);

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

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

// for SD Card
  pinMode(CS_pin, OUTPUT);

  // SD Card Initialization
  if (SD.begin())
  {
    Serial.println("SD card is ready to use.");
  } else
  {
    Serial.println("SD card initialization failed");
    return;
  }
 sdcard_file = SD.open("PMS001.csv", FILE_WRITE);
  if (sdcard_file)
  {
  sdcard_file.print("time (ms)");          // milliseconds since start
  sdcard_file.print(", ");      //write a commma
  sdcard_file.print("PM2.5");                          //write data to card
  sdcard_file.print(", ");      //write a commma
  sdcard_file.print("PM10");                          //write data to card
  sdcard_file.println(", ");                          //write a commma
  sdcard_file.close();                                //close the file
  }
  // if the file didn't open, print an error:
  else {
  Serial.println("error opening file");
  }
```

```
  delay(1000); //wait a second
}
// this next section tells the arduino what structure the incoming data has so it can grab the right
bytes

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_100um;
  uint16_t unused;
  uint16_t checksum;
};

struct pms5003data data;

void loop() {
  if (readPMSdata(&pmsSerial)) {
    // reading data was successful!
    uint32_t m = millis();
      Serial.print(m);
      Serial.print(" ");
      Serial.print(data.pm25_env);
      Serial.print(" ");
      Serial.print(data.pm100_env);
      Serial.print("\n");

    // write to SD Card

    sdcard_file = SD.open("PMS001.csv", FILE_WRITE);
  if (sdcard_file)
  {
  sdcard_file.print(m);          // milliseconds since start
  sdcard_file.print(", ");      //write a commma
  sdcard_file.print(data.pm25_env);                          //write PM2.5 data to card
  sdcard_file.print(", ");      //write a commma
  sdcard_file.print(data.pm100_env);                          //write PM2.5 data to card
  sdcard_file.println(", ");                          //write a commma
  sdcard_file.close();                          //close the file
  }
  // if the file didn't open, print an error:
  else {
  Serial.println("error opening file");
  }
  }
}

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;
}
```