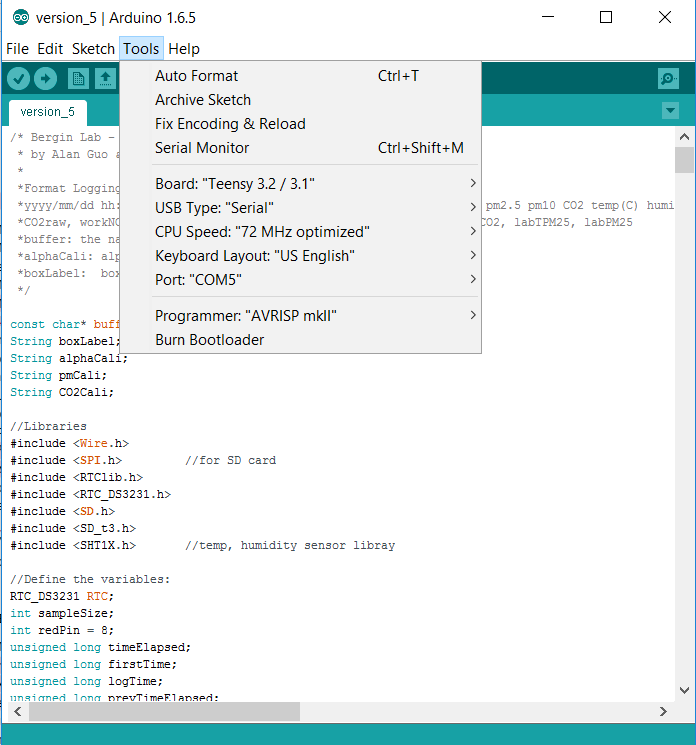
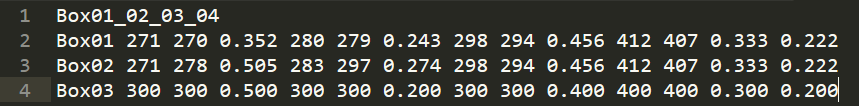
**GENERAL GUIDE LINE: STEPS TO SET UP TEENSY SENSOR PACKAGE IN LAB**

1. Make sure all parts are connected to the PCB.
2. Make sure the SD card has properly formatted CALI.txt and LAB.txt in it. (details included in the sections below)
3. Make sure SD card is properly inserted in.
4. Make sure Arduino, Teensy and all related libraries are installed on your computer:
   1. All the libraries are installed on the lab’s computer and can be simply copied and pasted onto any computer. Copy the entire folder /Documents/Arduino/libraries into the same Arduino folder on your computer and you should be set.
   2. If any library is lost, simply google the library name, “SHTIX.h” for example. The libraries are normally open-sourced in GitHub. To install library, you need to copy the downloaded folder into /Documents/Arduino/libraries.
5. Plug in the USB wire into Teensy. (The Teensy will blink a little if it’s properly connected)
6. Go to the Duke Box Vera’s Arduino code folder, download the “version\_6.ino” file. Open the file in Arduino. Arduino will ask you if you want to put the file in a folder, say yes, and it will automatically do it for you.
7. Go to Tools. Make sure the settings are as followed: “Board: Teensy 3.2/3.1”, “USB Type: Serial”, “Port: COMx” (x can be any number depending on the Teensy). Under “CPU speed”, it’s commonly auto selected by the software. It’s safe to just use default there, but if the default doesn’t work for some reason. A debugging trick is to change the CPU speed to a lower or higher rate.
8. Compile and upload the codes. A teensy window will pop up and ask you to press button on Teensy. Press the button and the teensy will start running.
9. Solid LED means the teensy is working. The teensy will blink rapidly 5 times every 60s when it’s logging data. The LED will keep blinking if either SD card is out or the required files are not present.



**HOW TO ENTER ALPHASENSE CALIBRATION INTO CALI.txt**

1. Make sure there’s a txt file named *CALI.txt* in the SD card;
2. The first line in the CALI.txt file contains the boxLabel formatted as “Box0x\_0x\_0x\_0x”, an example would be “Box01\_02\_03\_04”. The numbers are for Teensy, Alphasense, PM and CO2 respectively. Right now, the PM and CO2 labels are not used for calibration in the Teensy codes. Only the alpha sense label, the second number, is used to do factory calibration in the code. But the entire boxLabel “Box0x\_0x\_0x\_0x” will be printed to the output file.



1. Apart from the boxLabel, each line contains all alphasense calibrations for one specific sensor box. The data is formatted as such “BoxNumber WEZeroNO AEZeroNO sensitivityNO WEZeroCO AEZeroCO sensitivityCO WEZeroNO2 AEZeroNO2 sensitivityNO2 WEZeroO3 AEZeroO3 auxNO2O3 sensitivityO3”, an example would be “Box01 271 270 0.352 280 279 0.243 298 294 0.456 412 407 0.333 0.222”;
2. All calibration values can be found on datasheet provided by the factory. Use TOTAL WE Zero and TOTAL AE Zeros.
3. Make sure every two values are separated by exactly on space between them.
4. If you were to add new calibrations, put your new lines in similar format.
5. To upload any codes, make sure the *CALI.txt* file is present with the correct calibrations in the SD card. Also make sure that the boxLabel in the CALI.txt is correct.

**HOW TO READ DATA FROM SD CARD**

1. Simply pull out SD card. Be careful not to disrupt any other part of the circuit;
2. Read SD card using a microSD card reader. The logged data will be in the file called *TEENSY.txt*;
3. You will notice now the LED light start and keep blinking. This means that SD card is not present and microcontroller is unable to write into SD card;
4. Replace the SD card or reposition it;
5. Pull out and re-plug in the power. This restart the microcontroller. Wait until the LED stops blinking and turns solid on;
6. To make sure that data is properly logged into the SD card, wait for ~60s. If you see the LED blink rapidly five times, you are all set!
7. Note that to restart Teensy after SD card is pulled out, you will have to put back the SD card, disconnect and reconnect the power to ensure that the microcontroller is starting over again and all initialization parameters are correct.

**HOW TO PUT IN LAB CALIBRATION DATA**

1. The Teensy code supports linear calibration data input. To make it easier for user to put in calibration data, there’s a LAB.txt file in the SD card with calibrations.
2. Each line in the text file represents one linear calibration for a set of data. It’s formatted as such “SensorName slope intercept”. For example, “Alphasense 1.5 3”.
3. The order of the sensor calibration in the text file matters, so please don’t intend to change the order of the sensor calibration, just change the numbers.
4. The default calibration is slope=1, intercept=0.
5. The input can be up to four sig. fig.
6. Before testing, make sure the *LAB.txt* file is present in the SD card. Make sure it’s named exactly “LAB.txt”.

*Coding tips: if for any reason, you need to change the linear calibration to a different form of calibration(exponential, polynomial, etc). Go to the LabCalibration() function in the Teensy code. myData is an array storing what’s read out of the LAB.txt file. It currently has a length of 18. The current algorithm is, in the txt file, if two words/numbers are separated by either a space or a line change, they will be recognized as two separate inputs and stored separately. That’s to say, for example, an input like “NO 1.5 0” will store “NO” into myData[0], “1” into myData[1], etc. Make sure only use numbers (float or double) in the equations.*

**

*Additionally, orders of these calibration matters. It should be NO, NO2, O3, CO2, TPM25, PM25, each occupying a line. DO NOT change the order of these calibrations. If you need to add another parameters, make sure to understand the algorithms in LabCalibration() before you add anything.*

**CODE STRUCTURE EXPLAINED**

The code was implemented with modularity. Modular programming is a software design technique that emphasizes separating the functionality of a program into independent, interchangeable modules, such that each contains everything necessary to execute only one aspect of the desired functionality. In the main loop(), modular functions are called to collect data from each sensors, read calibrations from SD card, do calibrations, calculate averages and write into SD card.

Modularity makes debugging and troubleshooting easier. In the case when one functionality is not working as expected, users can just go to that specific part of the code to troubleshoot.

Below is pseudo-code that outlines Teensy code structures:

*Setting up boxLabels and libraries.*

*Define global variables (separated by functions)*

*void setup () {*

*Setup sensors;*

*Prepare SD login strings;*

*Setup timer;*

*Factory Calibration for Alphasense;*

*Turn off LED for a sec to mark end of setup;*

*}*

*void loop() {*

*bool getFlag = getPMValues(); //Store PM values into global variables*

*if (getFlag) { //getFlag=1 if checksum proves PM values correct, else getFlag=0*

*TimePrint(); //Print time to serial monitor*

*PMPrint(); //Print PM values to serial monitor*

*getCO2Values(); //Collect CO2 values and print on serial monitor*

*getSHT1X(); //Collect temperature and humidity and print on serial monitor*

*getAlpha(); //Collect NO, NO2, O3 values with fac calibration*

*TimeElapseCalculation(); //Collect and print time elapsed on serial monitor*

*if (60s from data log last time) {*

*AverageCalculation(); //Calculate minute average for all data*

*LabCalibration(); //Send minute average through lab calibrations*

*Complete data string using the global variables;*

*Log data into SD card and check if log in is successful;*

*ReSet(true); //Reset minute averages*

*}*

*}*

*}*

**ADDITIONAL INFOS ON PLANTOWER PMS3003 AIR QUALITY SENSOR S**

*Useful references: http://www.dfrobot.com/wiki/index.php?title=PM2.5\_laser\_dust\_sensor\_SKU:SEN0177*

1. **How to connect the Plantower sensor to microcontroller?**

|  |  |  |
| --- | --- | --- |
| PIN1 | VCC | Connected to +5V power supply |
| PIN2 | GND | Connected to ground |
| PIN3 | SET |  |
| PIN4 | RXD | Connect to Serial Port RX1 on Teensy |
| PIN5 | TXD | Connected to Serial Port TX1 on Teensy |
| PIN6 | RESET |  |
| PIN7/8 | NC |  |

1. **How does the communication protocol work?**

The Plantower PMS3003 gives 24-byte output that follows the protocol below. The first 4 bytes are always 42, 4D, 0 and 14. The next 12 bytes are the 6 PM measurements, the next 6 are not defined and the last 2 are the checksum.

|  |  |  |
| --- | --- | --- |
| Start Character 1 | 0x42(fixed bit) | |
| Start Character 2 | 0x4d(fixed bit) | |
| Frame Length 16-byte | Frame Length = 2\*9+2 (data+check bit) | |
| Data 1 | concentration of PM1.0, ug/m3 | |
| Data 2 | concentration of PM2.5, ug/m3 | |
| Data 3 | concentration of PM10.0, ug/m3 | |
| Data 4 | Internal test data | |
| Data 5 | Internal test data | |
| Data 6 | Internal test data |
| Data 7 | the number of particulate of diameter above 0.3um in 0.1 liters of air | |
| Data 8 | the number of particulate of diameter above 0.5um in 0.1 liters of air | |
| Data 9 | the number of particulate of diameter above 1.0um in 0.1 liters of air | |
| Data 10 | the number of particulate of diameter above 2.5um in 0.1 liters of air | |
| Data 11 | the number of particulate of diameter above 5.0um in 0.1 liters of air | |
| Data 12 | the number of particulate of diameter above 10.0um in 0.1 liters of air | |
| Data 13 | Internal test data | |
| Check Bit for Data Sum | Check Bit = Start Character 1 + Start Character 2 + ...all data | |

In addition, a checksum function *int checkValue(uint8\_t thebuf[24], int leng)* was implemented to check validity of the PM output. The output int=1 when the datastream is checked to be valid, else int=0.