I2Cpytools – Manual (version 0.2)

by ullix, April 2018

Software and documents available for download at
https://sourceforge.net/projects/i2cpytools/

I2C sensors are low-cost, readily available devices to measure environmental variables, like temperature, humidity, air pressure, light intensity, and others more. Other modules are e.g. LED matrices and other types of displays.

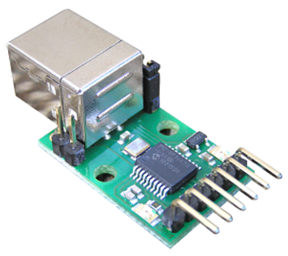
However, while there are plenty of tools for using them with microcomputers like Adafruit and Raspberry Pi, few are available that allow controlling an I2C sensor or module from a PC or laptop.

Fortunately there are hardware devices available, which allow the connection of an I2C sensor to the PC via a USB port. I will call these devices dongles.

Hardware Overview - Dongles

I2Cpytools currently supports **three different dongles**, which can be connected simultaneously. These dongles are:

- **ELV USB-I2C** from ELV Elektronik AG https://www.elv.de/
- **IOW24-DG** (IO-Warrior24 Dongle) from Code Mercenaries https://www.codemercs.com/de
- **USB-ISS** from Devantech https://www.robot-electronics.co.uk/htm/usb_iss_tech.htm

The **ELV** dongle	The **IOW** dongle	The **ISS** dongle
		

Hardware Assessment – Dongles

The dongles are in a price range from 30 to 40€. A hardware consideration is their support of both 5V and 3.3V, or only one of them. And then there is the issue of programming them.

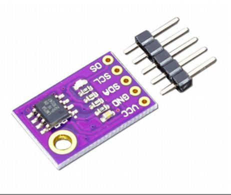
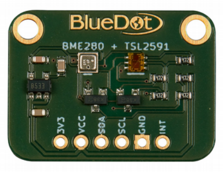
- The ELV dongle is **by far the easiest to program!** It has three connectors for I2C devices. However, while this may sometimes be convenient, it is of limited value as they are connected internally. This is the same bus with three connectors, not three different buses!You could just as well connect several I2C devices externally, e.g. on a bread board or solder them together, as you would do for the other two dongles.

It handles 5V devices; for 3.3V some soldering and external resistors are required (https://www.elv.de/faq/usb-i2c-interface-kompatibilitaet-mit-3-3-v-slave.html).
The ease of programming makes this my preferred dongle!
Price paid: 41€.

- The IOW is the **most complicated to program**. Part is due to the fact that you must link the Python code to a C-library, but then you also need more lower-level programming sub steps for each command.
  By default it works with 5V devices, but can be modified to work with 3.3V. This requires some soldering including zener diodes and resistors.
  (https://www.codemercs.com/downloads/iowarrior/IOW24-DG_Datasheet.pdf)
  Price paid: 32€.
- The ISS is also easy to program – in principle. But it has a ‘feature’ – I am inclined to call it **a bug** – that makes it unnecessarily difficult. It deviates from the standard readout practice of I2C devices, and requires a work around. It makes communicating with I2C devices further complicated by requiring different commands for 1 and 2 byte registers of the I2C device. It will fail if there ever is a 3 byte register; see details as comments in the code. It already failed to talk to the LED matrix with code that worked flawlessly with the other dongles.
  It should work with 3.3V devices by simply removing a hardware link (not tested).
  Due to the buggy firmware it is **not** recommended as an I2C dongle!
  Price paid: 28€

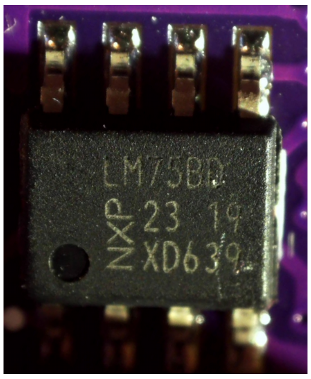
Hardware Overview – Modules

Each I2C module may carry one or more I2C sensors or other devices on it. Three modules, two with one and one with two devices, were used. Eventually, all worked on all dongles, either all connected to one single dongle, or all three dongles activated with each being connected to only a single module :

LM75B Module with single sensor for temperature. Moderate resolution and accuracy, but very easy to program. Possibly having a firmware bug, see below Price paid: 6€	
BlueDot BME280 + TSL2591 Two sensors on one module, both with complex programming: **BME280** for temperature, barometric pressure, relative humidity **TSL2591** for light intensity in visible and infrared region Price paid: 22€	
LED Matrix 8x8 HT16K33 64 red LEDs in an 8x8 matrix This particular one with a defect: the columns are shifted by one row so that the bit #8 LED lights up where the bit #0 LED should be! Price paid: 3€	

Hardware Assessment – Modules

LM75



The LM75 type sensors measure **temperature**. They are available from several manufacturers. They come in types from LM75 (9 bit resolution) to LM75A to LM75B (both 11 bit resolution). All have the same (poor) accuracy. There also appears to be a firmware bug (see below) at least in my LM75B variant from NXP (see picture).

I would not use this sensor for any serious monitoring (e.g. weather station). But it is so easy to program that I'd recommend it as a **starter device** for anyone beginning working with I2C devices.

BME280

This sensor from Bosch provides **temperature**, **pressure** and **humidity** data in very good precision and good accuracy.

The temperature and humidity values are in good agreement with other analog and digital devices in my home, though none are calibrated against a good reference. The pressure data are in very good agreement with data from a Marine-GPS with pressure sensor, and follow well the local weather reports. All data come out smooth with no jitter. Measurements, even in slow setting, are faster than I need them.

This seems like the gold standard for any in-house or weather monitoring.

Programming this device was a real challenge, though this is solved now!

BlueDot BME280 + TSL2591

My BME280 is mounted on a board together with a TSL2591 light sensor. The board is produced by BlueDot. It is convenient if you want light as an additional environment variable.

TSL2591

This sensor measures light in two channels: one in the **visible light** range, one in the **infrared** range. As the visible light photo diodes also detect infrared (test it: take your digital camera and look at the infrared LED of your TV remote), you could use the IR signal of the sensor to correct the visible light sensor for the IR contribution (not yet implemented in **I2Cpytools**).

The sensor has an extreme dynamic range of 600 million : 1, which it achieves with a variable gain of 1 to 9876 plus an integration time from 100ms to 600ms. **I2Cpytools** is programmed to keep the integration time constant at 600ms, and to auto-adjust the Gain to obtain the highest possible resolution, and then to normalize the result.

The results were verified to be consistent over all gains and integration times.

The intensity results vary over such a wide range that it is best to plot not the intensity, but rather the logarithm of it. This is done by the scaling option within the configuration of the companion plotting program **pytoolsPlot.py**.

LED Matrix 8x8 HT16K33

The LED matrix has 64 LEDs in an 8x8 array. The LEDs are quite bright in their highest setting (fully lit module draws ~170mA) but can be dimmed to more pleasing levels.

Programming is simple, although still a bit more demanding than the LM75 sensor. So this is another good I2C **starter device**. The problem in my device is a manufacturing defect that shifts the columns by one row so that the bit #8 LED lights up where the bit #0 LED should be! A workaround is programmed in **I2Cpytools.**

Although programming is simple, and was easily implemented in the ELV and IOW dongle, it took some time to understand what is wrong in the ISS dongle. The peculiarities of this dongle's firmware required a non-standard way of programming, certain normal, straight-forward programming steps simply don't work with this dongle!

Again, the ISS dongle is not first choice for I2C applications!

Software Overview

I2Cpytools is developed on Linux using Python3 (3.5.2) scripts. It will therefore run wherever Python3 runs. It runs in a terminal.

Exception: The IOW supporting part of I2Cpytools depends on a library, which currently is specific for Linux! Therefore the IOW dongle in this version of **I2Cpytools** will run only on Linux systems! Windows libraries are available and could be implemented. See the Windows SDK here: https://www.codemercs.com/downloads/iowarrior/IO-Warrior_SDK_win.zip

The single-keypress-commands rely on the curses module, which is not available on Windows. Therefore this part will also not work on Windows. But these functions are for convenience only, they are not essential for running the program.

Logging

I2Cpytools runs as data logger. It reads all sensors, then prints the data to the screen, saves the data in a log file in CSV (Comma Separated Values) format, and lights some LEDs on the matrix display as rough indication of temperature, pressure and humidity values. After a user defined cycle time (default: 30 sec) this cycle is repeated.

With CTRL-Z the current cycle time can be shown and changed (≥ 0 sec).

The log file will have the format (lines starting with '#' are comments)

```
#   Index, Date&Time,             T,      P,        H,      T,     L
    12990, 2018-03-14 06:17:22,  21.89,  1004.59,  31.51,  23.08,  103.06,
    12991, 2018-03-14 06:17:25,  21.90,  1004.62,  31.47,  23.08,  102.96,
    12992, 2018-03-14 06:17:28,  21.89,  1004.62,  31.48,  23.09,  103.13,
```

Starting

Start I2Cpytools in a terminal with:

```
path/to/I2Cpytools
```

The data will be saved into a log file named ‘Sensor-<Date> <time>.log’ in the subdirectory ‘data’ of the program directory.

Your can also specify the log file name (any extension ok) on the command line by starting with:

```
path/to/I2Cpytools   path/to/mylogfile
```

Command line options for starting - Help

To get Help info, simply start the program with:

```
path/to/I2Cpytools   -h
```

This will show:

```
Usage:  I2Cpytools [Options] Datafile

Options:
    -h, --help          Show this help and exit
    -V, --Version       Show version status and exit
    -P, --Ports         Show available serial ports and exit
    -c, --config name   Set the plotting config file to name;
                        name is the config file incl. path
                        Default is 'cfg/pytoolsPlot.cfg'
    -l, --last N        Plot only the last N records

Datafile                File must be of type CSV
                        (Comma Separated Values)
                        If data file is not in same
                        directory as program, a relatve
                        or absolute path must be given:
                        e.g.: path/to/csvfile4plotting
```

Command line options for starting – Version

To print the versions of **I2Cpytools** and the active Python version, type:

```
path/to/I2Cpytools   -V
```

This will show e.g.:

```
Version status:
   I2Cpytools     : I2Cpytools-0.2
   Python         : 3.5.2 (default, Nov 23 2017, 16:37:01) [GCC 5.4.0
20160609]
```

Command line options for starting - Serial Ports

Both the ELV and the ISS dongle use a USB-to-Serial converter, which will make the device show up under port ‘/dev/ttyUSB0’ (ELV) or ‘/dev/ttyACM0’ (ISS). Under Window they both show up as a ‘COMX’ port (X=0,1,2,…). To get the available serial ports, start **I2Cpytools by:**

```
path/to/I2Cpytools   -P
```

for an answer like:

```
Available Serial Ports:
   /dev/ttyUSB0 - ELV USB-I2C-Interface
   /dev/ttyACM0 - USB-ISS.
```

Command line options for starting – Configuration File

Plotting of the data will be done with the included plotting tool **pytoolsPlot.py**, see chapter below**.** A configuration file for this plot can be defined by:

```
path/to/I2Cpytools   -c path/to/myconfigurationfile
```

Command line options for starting – Last Lines

When the log file becomes very large (>> 10000 lines) plotting may take an inconveniently long time. You can limit plotting to the last N lines by -l N (-l like in last), e.g. for the last 5000 lines:

```
path/to/I2Cpytools   -l 5000
```

Stopping

With CTRL-C the program can be stopped cleanly; all open files and devices will be properly closed and terminated.

Running I2Cpytools

Configuration

You can run any one dongle alone, or in combination with another up to all three simultaneously. Each one will span its own I2C bus. A sensor or module can be activated on any bus, but not on more than one. (The two sensors on the combo module BlueDot obviously must be activated for the same bus).

This activation is currently done in the Python code of **I2Cpytools**. Look at this segment (near line 100): Modify only the “if 10:” to “if 0:” or vice versa, nothing else. In this example all 3 dongles are activated, and on ELV the LED matrix is active, on IOW the sensor LM75B, and on ISS the combo BME280 and TSL2591.

```
#####################################################################################
# BEGIN user activation BEGIN user activation BEGIN user activation BEGIN
#####################################################################################
# activate any or all dongles and connected sensors
    if 10:
        print("\nactivating ELV USB-I2C dongle @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@")
        glob.elv = ELV.ELVdongle()
        #glob.elv.ELVreset()        # do a dongle software reset (takes 2 sec)
        #glob.elv.ELVshowInfo()     # show dongle info available with '?'
        #glob.elv.ELVshowMacro()    # show dongle content of the macro memory
        # activate the sensors connected to ELV
        if  0: glob.LM75        ["dngl"]     = glob.ELVdongle
        if  0: glob.BME280      ["dngl"]     = glob.ELVdongle
        if  0: glob.TSL2591     ["dngl"]     = glob.ELVdongle
        if 10: glob.HT16K33     ["dngl"]     = glob.ELVdongle
    if 10:
        print("\nactivating IOW24-DG dongle @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@")
        glob.iow = IOW.IOWdongle()
        #glob.iow.IOWshowInfo() # show IDs on hard- and software, versions, etc
        # activate the sensors connected to IOW
        if 10: glob.LM75         ["dngl"]     = glob.IOWdongle
        if  0: glob.BME280       ["dngl"]     = glob.IOWdongle
        if  0: glob.TSL2591      ["dngl"]     = glob.IOWdongle
        if  0: glob.HT16K33      ["dngl"]     = glob.IOWdongle
    if 10:
        print("\nactivating USB-ISS dongle @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@")
        glob.iss = ISS.ISSdongle()
        glob.iss.ISSshowInfo() # show IDs on hard- and software, versions, etc
        # activate the sensors connected to ISS
        if  0: glob.LM75        ["dngl"]     = glob.ISSdongle
        if 10: glob.BME280      ["dngl"]     = glob.ISSdongle
        if 10: glob.TSL2591     ["dngl"]     = glob.ISSdongle
        if  0: glob.HT16K33     ["dngl"]     = glob.ISSdongle
#####################################################################################
# END user activation END user activation END user activation END
#####################################################################################
```

After any changes save and restart **I2Cpytools.**

Runtime Commands

The running program can be controlled with these commands:

CTRL-C	The program will be stopped cleanly. All open files and devices will be properly closed.
CTRL-Z	The current cycle time can be shown and changed (≥ 0 sec).
CTRL-\	The auto-plotting can be set:

- 0 : OFF mode:
  no plotting; if a plotting cycle had been started, it will be switched off
- -N : ONCE mode: (any negative number)
  show a plot, and then switch to Off mode
- N : CYCLE mode: (any positive number)
  show the plot and start a cycle with updates every N seconds
  (but not faster than the logging cycle)

When on Linux you can also give single letter commands (no return key needed). This is convenient in particular for long running data collections, but not essential (use small cap or large cap, except for p/P) :

p	(small cap) Plot the last N data if the last N option had been given, or all if not
P	(large cap) Plot all the data, ignore last N, even if given
m, M	Take a measurement
i, I	Provide info, like:

```
Info:
   Logfile                     : data/LongRun1.log
      File size (bytes)        : 57,547,274
      No of lines              : 316,901
   Configfile                  : cfg/I2Csensors.cfg
   Cycletime (sec)             : 30.0
   Graphcycle (sec) (0=OFF)    : 0
   Last records to plot        : 20000
   Dongle in use               : ELVdongle
      connected with           :     LM75
      connected with           :     BME280
      connected with           :     TSL2591
      connected with           :     HT16K33
   Dongle NOT in use           : IOW24-DG
   Dongle NOT in use           : ISSdongle
```

q, Q	Quit **I2Cpytools** (alternative to CTRL-C)
h, H	Help (same as the command line option -h)
v, V	Versions (same as the command line option -V)

Runtime Printouts

I2Cpytools prints out quite a bit of text, which will be limited to Debug info in later versions. Any “good news” (found dongle, found sensor xyz) will be printed in green, “bad news” (Errors, Warnings) in yellow.

After every measurement cycle the results are being saved to the log and printed out. The first line (starting with ‘logtext’) printed in blue gives the current results, the second line (starting with ‘average’) printed in green gives the average over the last 10 cycles (or all cycles if there were fewer than 10). This is to provide some assistance for the interpretation, as the BME280 in particular is very sensitive and quickly responding. E.g. your breathing near the sensor will impact the humidity value, and other, like the LM75, may show too much jitter.

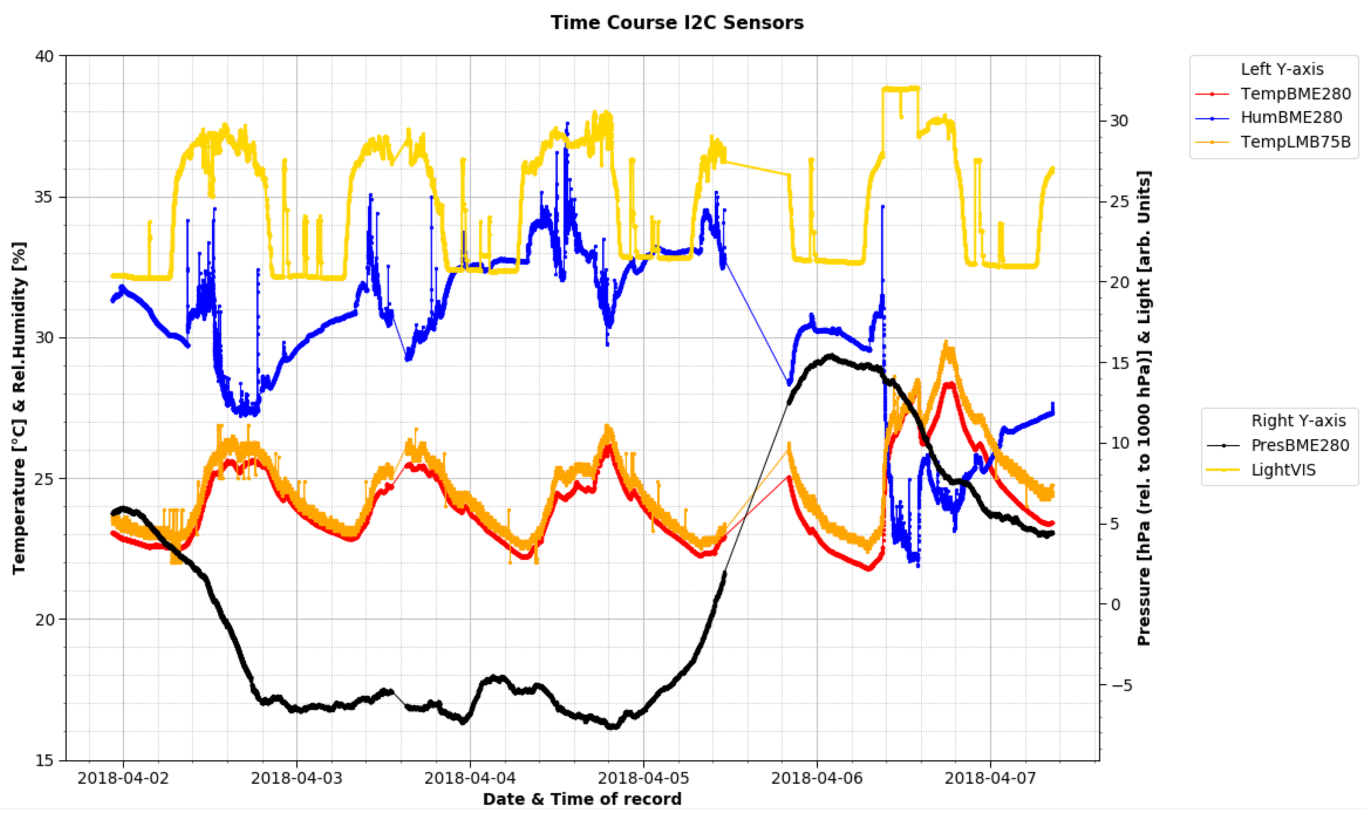
```
                    DateTime,        T,           P,        H,   T_LM75B
logtext: 2018-04-06 14:57:10, 26.2900,   1010.2773, 25.5919,   27.1250
average: 2018-04-06 14:57:10, 26.3,      1010.3    , 25.6    ,   27.1
```

Plotting

Included in the **I2Cpytools** package is the plotting tool **pytoolsPlot.py.** This actually is stand-alone software, which allows to plot data from any file in CSV format. It is also a Python3 script and will run on all operating systems with Python3 support.

This software has options for configuration of the layout. See the accompanying manual file **pytoolsPlot-Manual.pdf** for details.

I2Cpytools uses this plotting tool **pytoolsPlot.py** by default to produce plots such as shown in the next figure. See chapter Runtime Commands above for the commands (CTRL-\, or p, or P) to trigger plotting.



Plotting from a second terminal window

While it is convenient to trigger the plots from within **I2Cpytools** for a quick view, sometimes you want to test different configurations for plotting and not disturb your ongoing data collection.

This is possible by calling **pytoolsPlot.py** from a second terminal window – while **I2Cpytools** continuous to run in the first – with:

```
path/to/pytoolsPlot.py /path/to/logfile
```

To apply different configurations you can use command line options, like:

```
path/to/pytoolsPlot.py -c /path/to/configfile -l 5000 /path/to/logfile
```

See the accompanying manual file **pytoolsPlot-Manual.pdf** for details.

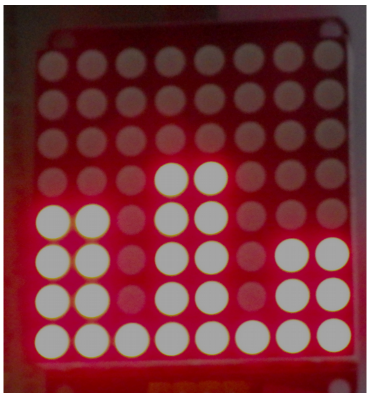
Discussion of Results

A typical Time Course result obtained with all sensors presented here is shown in the above figure. This plot uses the configuration file **I2Csensors.cfg**, also included in the package, as an example.

Pressure is the black curve, shown in units 'hPa minus 1000' on the right hand Y-axis. The bad-weather days with low pressure, followed by 2 days of sunshine in high pressure are correctly shown.

Humidity is the blue curve (left Y-axis). The many spikes are real data, simply denoting the sensitivity of this sensor. Little disturbances near it – like your breathing, opening windows – will result in spikes.

Light Intensity is the yellow curve. It uses the scaling math formula 'log(col)+25' to plot the data, which means to take the natural logarithm of the data, and adds 25 to shift their plot upwards on the figure (right hand Y-axis). It shows daytime light and scatter from clouds, plus some artificial lighting in the dark hours.



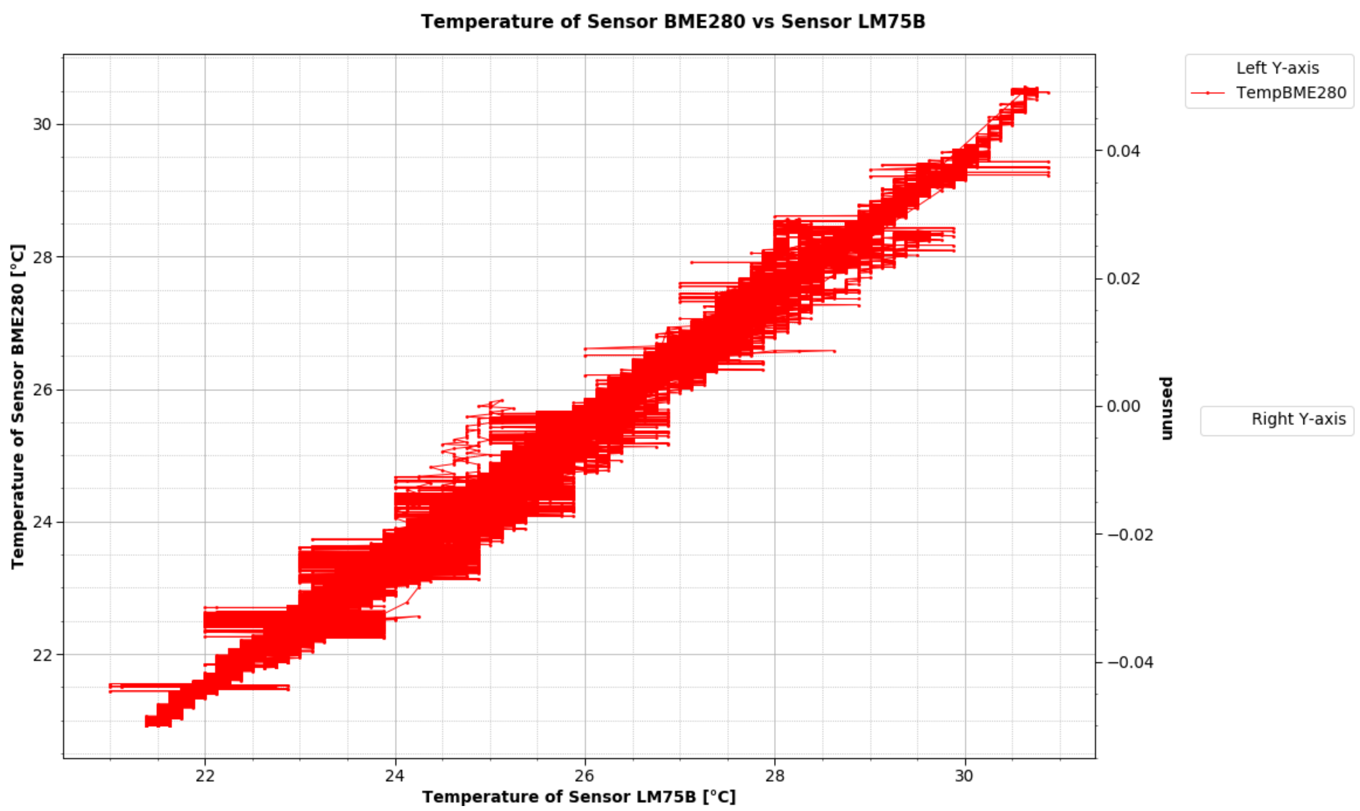
Temperature are the red (from the BME280 sensor) and orange (from the LM75B sensor) curves (both on left Y-axis). The LM75B is on average 0.50°C higher than the BME280 – but I don't know which of the two is the more correct value.

LED Matrix - just to find some use for it, I let **I2Cpytools** plot 3 columns showing relative values for temperature, pressure and humidity (from left to right).

Bug in the LM75B firmware?

The temperature curves reveal an interesting issue: The red curve (BME280) is a thin, smooth line with very little jitter, the orange curve (LM75B), however, is a broader line due to the jitter in the signal. This is expected, given the much lower resolution of the sensor. But there are also numerous up-and-down spikes visible on the orange curve. On first glance they look like electrical disturbances, but after further evaluation this is not the case.

I plotted the BME280 temperature on the left Y-axis versus the LM75B temperature on the X-axis, simply by modifying the configuration file (present in the package as file **TempVsTemp.cfg**). The result is shown in the next figure:



The difference of 0.50°C between the two sensors is visible, as is the jitter in the LM75B signal. But in addition one sees the horizontal bars, which are not located randomly, but always as full integer numbers on the LM75B temperature (at 22, 23, 24,… degrees). And they all span from -1° to +0.875°C (in the plot window you can pan and zoom the plot to see the numerical details).

There is a strange relationship to how the LM75B sensor reports its data:

When asked for a temperature value, the sensor returns 2 bytes. The first one is the MSB and has 8 bits of full degrees (22, 23, 24,...°C), the second one is the LSB, and of it only the top 3 Bits are relevant. (Note: actually the readout is in Two's Complement format, but this matters only for negative Celsius degrees.) The three LSBit give the fractional degrees in units of 0.125°C, hence ranging from 0.000°C … 0.875°C.

So, it appears that the LM75B has a defect that comes into play whenever its temperature goes through a full integer value of the temperature, or in other words, when the fractional part becomes zero for all three bits.

I would like to hear if anyone else has observed this behavior!

Package

The content is:

- **I2Cpytools** and required Python scripts
- **I2Cpytools-Manual.pdf**
- plotting tool **pytoolsPlot.py** and its configuration default file **pytoolsPlot.cfg**, plus example files **I2Csensors.cfg** and **TempVsTemp.cfg**
- **pytoolsPlot.py-Manual.pdf**
- data file **segment.log**, a subset of what was used here