Program for measurement and calibrating data from observatory variometer

Albershteyn Andrey
alberand@fel.cvut.cz

January 13, 2016

Contents

1 Description

This program is used for data calibration and saving it. Data is received from observatory variometer by the serial bus. Main purpose of this software is to be easy to use and correctly working on Raspberry Pi.

Sensor contains FPGA chip which create data stream and send it via serial port. Sensor is connected to voltage level converter and than to the Raspberry Pi's serial interface.

Program is separeted into 3 different processes. First 'main' process is main flow of program. It communicates with sensor and receivs raw data from it. Second process 'processor' is responsible for data calibration, third process just saves calibrated data to files. Processes are connected by two pipelines and they are independent, only in case of terminating main process 'main', other processes will be terminated.

This program have a few test function, for debugging. One of them is that program will indicate every received sample by changing state of one of GPIO pins. This pin can be choosen in configuration file, by default it's pin 4. Its can be used for confirming that all samples sent by serial port are received by Raspeberry Pi.

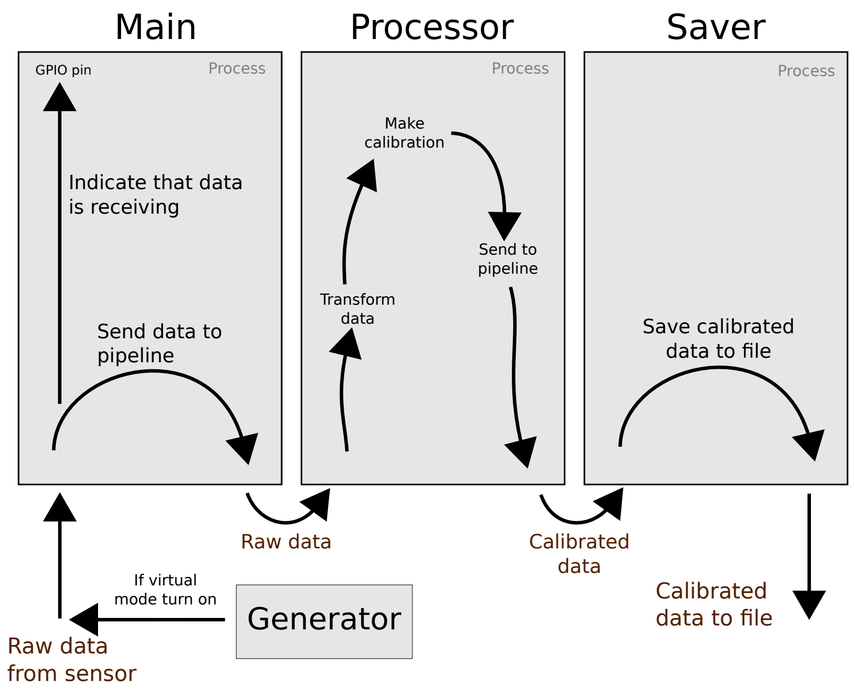
Other option is that we can run program in virtual mode and we don't need real sensor for it. In this mode program generate random data and use it as a input stream. It can be used for verification that calibration is working right way. And test program's functionality.

Also you can turn on logging for whole program cycle. Program contains information messages in different places to inform user about current performed task. This option is possible to turn on/off in configuration file.

Whole program is write with using only built-in python libraries. So it don't need third-party software. Only if debug mode is turned on (thus some of the GPIO pin is used for samples indication) it is necessary to install GPIO library to communicate with pins. Program is written in Python 3.2, so it should be compatible with all Python 3.* versions.

2 Program's structure

In this section we have program's structure. That is what is every file responsible for and how it connected.



Files' list:

1. **main.py** - This file is main input point to the program. Runs as a separated process and perform data reading from serial port and than send it to pipeline to following calibration.

2. **saver.py** - contains only one function, which is run as a separated process and perforn data saving.
3. **calibration.py** - contains a few utility function, which are used in processor for data calibration and some data transformation.
4. **processor.py** - contains one function, which is run as a separate process. Perform data calibration and send calculated value to pipeline to saving process.
5. **reader.py** - this file contains functions for serial communication setup and functions for communication with device.
6. **generator.py** - contains class, which is used as a generator of random numbers. It's used only in debug mode, when we don't have real sensor we use this generator to test program functionality.
7. **settings.py** - this file contains two classes which are responsible for program and calibration settings (next section).

3 Configuration settings

In this section we have description of configuration file. Program have two configuration classes. One for program configuration (where to store data files, turn on/off debug mode, serial communication settings etc.), second class is responsible for data calibration (offsets, sensitivitiy etc.)

Program configuration:

1. **samples** - number of sampels received per second.
2. **debug** - Turn on or turn off debug mode. Thus, program will indicate every sample on debug pin by sending impuls and all program's action will be logged.
3. **debug_pin** - GPIO pin which is used to indicate samples receving. Thus, this pin will change his state before and after data sample.
4. **path** - path where program will be store all files.
5. **port** - port for serial communication. Sensor should be connected to this port. By default port is '/dev/ttyAMA0'.
6. **baudrate** - serial communication speed. By default, speed is set up to 115 200.
7. **timeout** - timeout for serial communication. By default, value is 3.
8. **start_cmd** - string command, which is send to sensor to start it. Starts data stream.

9. **stop_cmd** - string, which is send to the sensor to stop it.
10. **file_name_format** - string, which define how filename will looks like.

Calibration settings:

1. **comp** - the compensation field
2. **ofs** - offsets
3. **sen** - sensitivity
4. **ort_mat** - the orthogonalization matrix

4 Functions' documentation

Description of functions

`calibration.`**`calibrate`**(*data*)

This function calibrate data and return numpy array. (np is numpy)

Parameters **`data`** – is list with 3 items. For example: np.array([123, 456, 789])

Returns np.array([111, 444, 777]) array with calibrated data

`calibration.`**`find_mean`**(*data*, *gauss*)

This function apply gauss filter to data and then calculate mean value. And multiply it by 2.

Parameters **`data`** – np.array([[H], [Z], [E], [T]]). H, Z, E and T are arrays

Returns np.array([[H], [Z], [E], [T]])

`calibration.`**`parse_data_string`**(*string*)

Cut string from sensors to separate values.

Parameters **`string`** – string in format like this '1234567123456712345671234567'

Returns [1234567, 1234567, 1234567, 1234567]

`calibration.`**`save_data`**(*data*, *path*, *suffix=''*)

Save data to path/%Y-%m-%d_suffix.txt

Parameters

- **`data`** – some data (For example: 123 123 123)
- **`path`** – path where to save file (For example: ./data/)

- **suffix** – suffix for filename (For example for suffix ‘original’ filename will be 2015-12-24_original.txt)

class generator.**Generator**

Bases: object

This class is used only in ‘Virtual’ mode. In this mode we are don’t have connected sensor and just generate random values for test program. So this class implements basic function of serial port.

read(*number_of_byte*)

This function return string in this format: 1234567;1234567;1234567;1234567; Numbers are just random.

Parameters **number_of_byte** – not used

Returns string

readline()

This function return string in this format: 1234567;1234567;1234567;1234567; Numbers are just random.

Returns string

write(*data*)

Just print received data.

Parameters **data** – string

processor.**process_data**(*pipeline*, *samples*, *path='./'*)

Process data from sensor. Accordingly get n samples and calculate average value from these samples. Then use Gauss filter and finally make calibration.

Parameters

- **pipeline** – pipeline where from this function will receive samples
- **samples** – number of samples per second
- **path** – path where we will save our files

reader.**init_communication**(*port='/dev/ttyAMA0'*, *baudrate=115200*, *timeout=None*)

This function create Serail communication object with default parameters:

Parameters

- **port** – serial port. By default ‘/dev/ttyAMA0’
- **baudrate** – By default 115200

- **`timeout`** – By default ‘None’

Returns pyserial object to communicate with device

`reader.`**`readline`**(*inpoint*)

Read one line from sensor.

Parameters **`inpoint`** – pyserial object

Returns line or False

`reader.`**`start_sensor`**(*inpoint*)

This function send start command to sensor (‘CN’).

Parameters **`inpoint`** – pyserial object

Returns True if start is successful otherwise False.

`reader.`**`stop_sensor`**(*inpoint*)

Send stop command to sensor (‘CS’). And return true if writing is successful.

Parameters **`inpoint`** – pyserial object

Returns True if stop is successful otherwise False.

`saver.`**`data_saver`**(*pipeline*, *path='./'*)

This function is run as a process and its save data getted from pipeline.

Parameters

- **`pipeline`** – pipeline
- **`path`** – path where to save data