BreatheIn2
Installation and usage instructions

This Python app reads live data from a (easily available and reasonably priced) CMS50D+ v4.6 finger oximeter. Displays a real time plot of the oxygen saturation (SpO_2) and pulse rate (PR) as well as the present values. Activates an external smartphone vibration via Bluetooth if SpO_2 descends below an adjustable minimum value. Disconnections/errors are handled gracefully.

This was developed to help with my idiopathic central sleep apnea (I stop breathing when having stressful dreams), so it is bedside-friendly. The smartphone goes under my pillow. Obviously, it is provided “as is”, without any implication that it may be useful to anyone’s specific case.

Optionally stores in disk at every heartbeat the SpO_2 and hearth rate as displayed by the oximeter, the time from the last heartbeat (RR time), max, min and average for the last heartbeat of the two pulse waveforms outputted (probably the red and infrared channels). An example matlab script is provided to read and analyze this information.

The present version was tested under Lubuntu 12.04 on Eee PC 4G and under Ubuntu 18.04 on a modern laptop. The smartphone was under Android Go.

Optionally (instructions provided) the oximeter can be modified to be powered from the data cable to avoid using batteries.

Installation

There are two installations to be made: on the bedside PC and on the smartphone. Instructions are also provided to modify the oximeter to be powered from the data cable.

PC

The Python2 scripts required are

```
BreatheIn2.py
cms50Dplus_thread.py
vibrate.py
```

These files can be installed on any folder and the program runs with

```
$ python BreatheIn2.py
```

The script `testBtooth.py` allows to test the connection to the smartphone.

On Lubuntu 12.04 two issues had to be solved:

```
$ sudo usermod -a -G dialout $USER # then log out and in again. This gives user access to /dev/ttyUSBx via dialout group
```

```
$ sudo apt-get install python-dateutil python-serial python-matplotlib # python dependencies
```

On Ubuntu 18.04 it runs without any modifications.

Don’t see a special problem to run this under Windows, but it was not tried.

Smartphone

Install the Scripting Layer for Android (*SL4A*) and Python for Android (Py4A) by following the instructions on https://pythonspot.com/sl4a-android-python-scripting/. To prevent eventual problems with versions of these apps the apks are also provided for manual installation.

Copy `vibroApp.py3` to `/storage/emulated/0/sl4a/scripts`. It should appear on the SL4A scripts list.

Pair the smartphone with the PC.

Obtain the bluetooth mac address of the smartphone (example: 00:04:61:02:AA:FF). On Android Go this is shown at the bottom of the bluetooth settings page.

`Run vibroApp.py3`. There should be two lines printed:

```
Cycling Btooth receiver
Accepting
```

On the PC run

```
$ python testBtooth.py 00:04:61:02:AA:FF # replace with actual mac
```

On the smartphone should appear timing ticks followed by alternate vibration and pause times in miliseconds. For example:

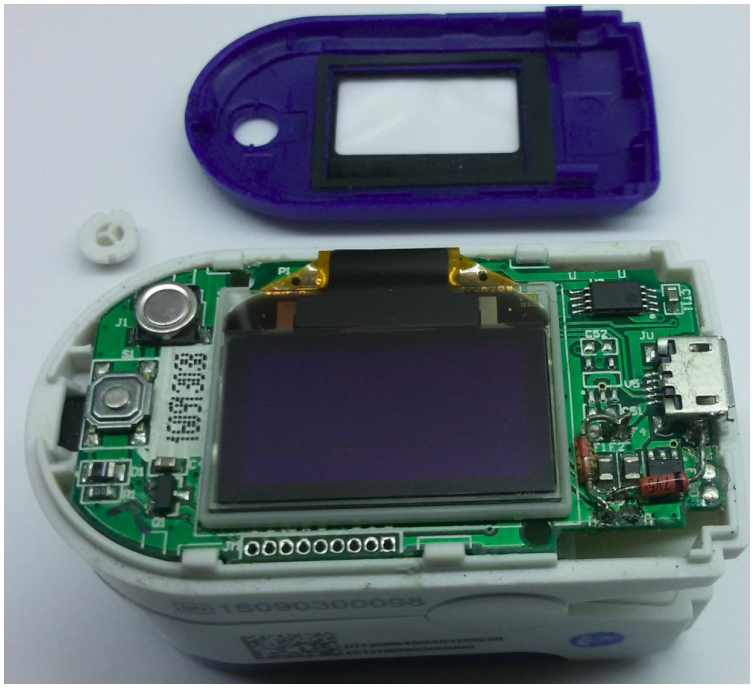
```
..............(50,50,200,0,0)
```

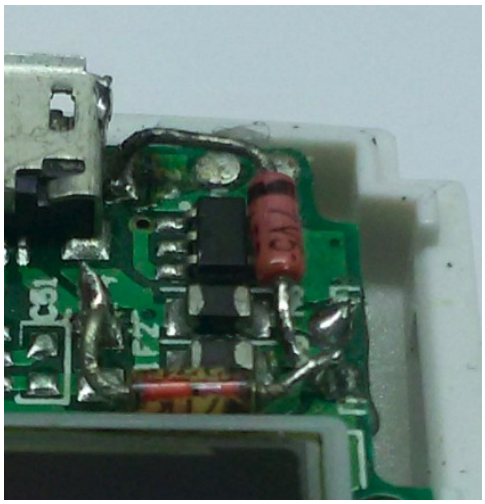
Several situations are tested and the script should return to the `Accepting` line.

Oximeter

To supply the oximeter from the data cable remove gently the top cover by inserting a thin screwdriver between the top blue cover and the white body and pulling the white part out on the locations of the 3 claws visible on the cover (next figure, left panel), while pressing the cover up.

Solder diodes (any kind of silicon diodes will do) between the USB power lines and the battery connection, as shown in the right-hand panel. Mind the polarity (strip on the body of the diode).





The battery can still be used, but not at the same time as the data cable.

Some hints about operating on batteries

Turning on the Record option in the oximeter auto shuts-off the display, saving battery.

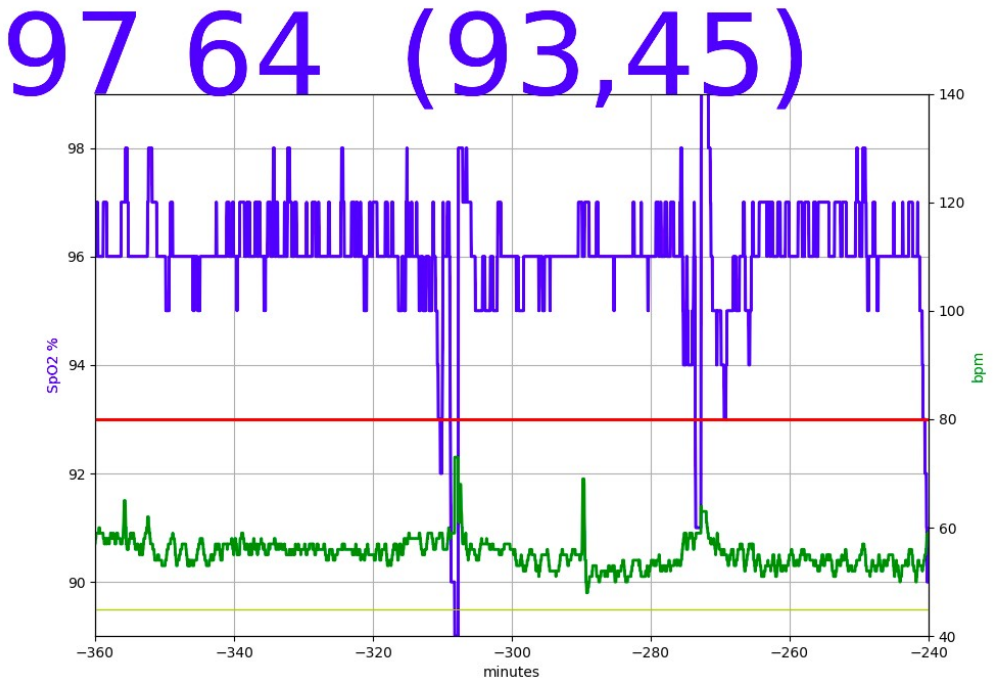
The battery alarm goes on at a very early stage, after a couple of hours, but the battery actually lasts at least up to 8 h with new NiMh batteries. Unfortunately this renders the alarm useless after a couple of hours as it will start to report a low battery. This is one of my motivations to write this program. The other is that I wanted a more progressive/smart alarm, also more friendly to other people in bed.

Usage

```
$ BreatheIn2.py -h
  usage: BreatheIn2.py [-h] [-a ALARM_MIN_SPO2] [-d DEVICE] [-o] [-m
  MACADDRESS]

  Download live data from a CMS50D+ oximeter. Store at every heart beat with
  time.

  optional arguments:
    -h, --help            show this help message and exit
    -a ALARM_MIN_SPO2, --alarm_min_SpO2 ALARM_MIN_SPO2
                          SpO2 minimal level. Default = 92
    -d DEVICE, --device DEVICE
                          path to CMS device. Example: /dev/ttyUSB0 (default)
    -o, --output
    -m MACADDRESS, --macAddress MACADDRESS
                          (bluetooth) mac Address of vibrator smartphone.
                          Example E0:48:D3:C2:39:83. Default None

  Commands:
    x = alarm off;
    space = toggle alarm on/pause (pause = 15 minutes);
    w = test vibration+beep;
    q or ctrl-c = quit;
    cvbnm = min SP02-1%;
    fghjk = min SP02+1%;
    arrow-up: beginning of the time plot;
    arrow-down: end of the time plot (present);
    arrow-left: 1h back in time;
    arrow-right: 1h forward in time;
```

These key commands should be issued with the focus on the display window.

The display window looks like:



The blue line is $Sp0_2$, the green line is PR, the red line is the minimum $Sp0_2$ (alarm level) and the yellow line the min PR.

The last (present) data is indicated on the top line, which has the structure:

`SP02 - blinking : at each data new data point (alive indicator) – PR – * (alarm indicator) – “ (data lagging indicator) – [ – minSpO2 – , – minPR – ] – ( – alarm countdown – )`

The color of this line indicates:

red – connection with the smartphone lost, beep will sound if PC sound is on (this is the only sound alarm);
black – alarm off
brown – alarm paused (countdown is displayed)

The line may also display `disconnected` (CMS is off or not connected), `finger off` , `searching` (CMS has raw data but not yet SpO2) or `no data` (normally transitory). For each of these conditions except `searching` there is a different vibratory alarm pattern.

The SpO_2 alarm is progressive: a single pulse of 50 ms * (min SpO_2 – present SpO_2). That is, 50 ms for each percentage point below min SpO_2. Maximum is 500 ms.

If the main program freezes, the `vibrate` thread will direct the smartphone to vibrate in groups of 3.

Some other functionalities

From `disconnect` or `finger off` states, returning the oximeter (on) to the finger enters the `searching` state which immediately stops the alarm.

If `finger off` state is reached within 5 s of a vibratory alarm a `pause` for 60 s is activated. Can be reactivated by `space`. This is to allow some time to recover the oxygenation without being bothered by the alarm.

The vibration function can be tested by removing the finger, which triggers an immediate double short pulse.

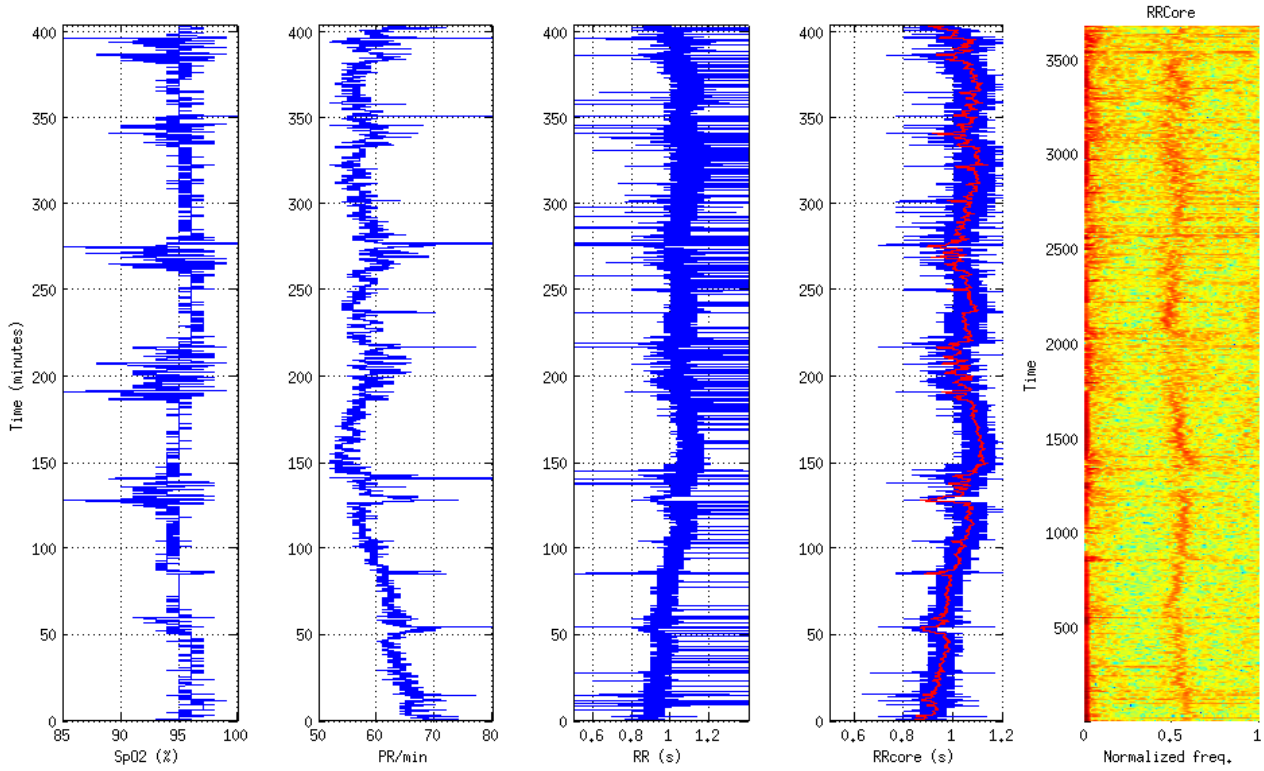
Example matlab analysis

The `BreatheIn2.m` script in the matlab folder reads the `.csv` file created by the app (-o option) and does some analysis.

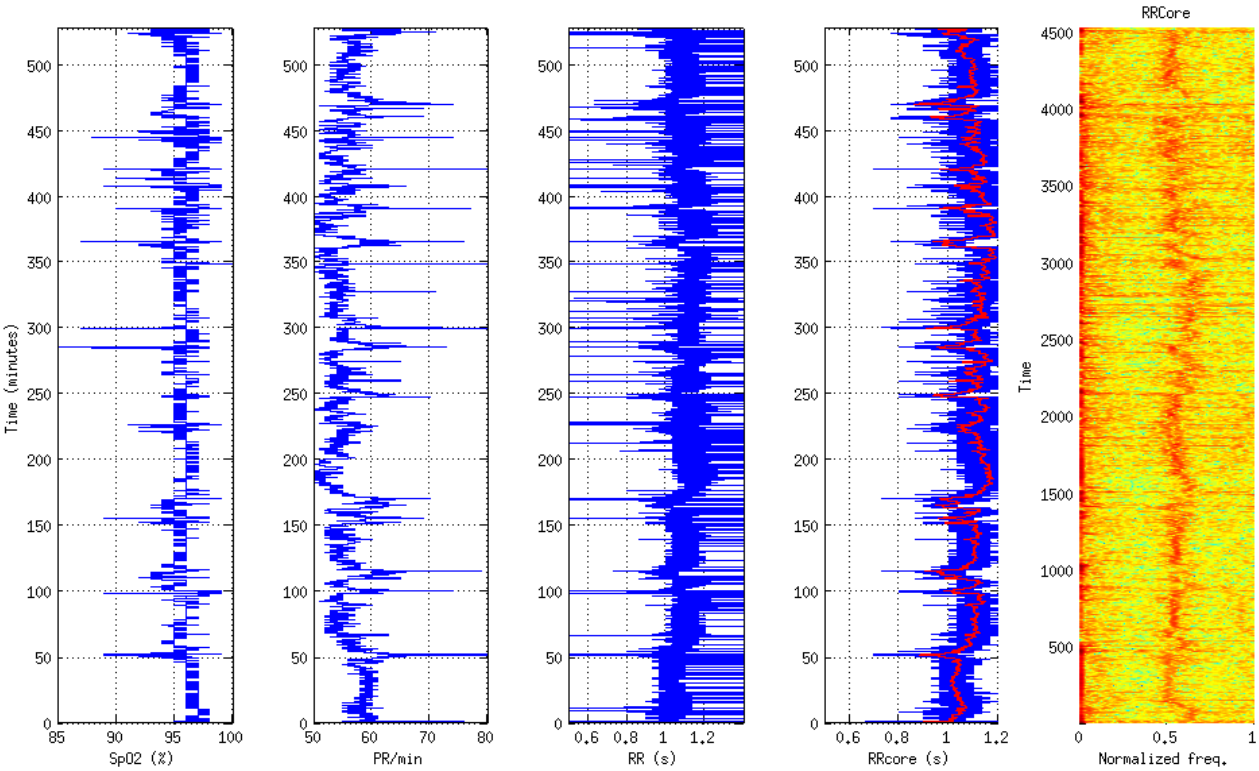
The plots in the images below are, from left to right, SpO_2, PR, the time between heartbeats (RR), the same with the outliers discarded (RRcore) and the smoothed curve (RRsmooth) and the spectrogram of RRcore-RRsmooth. The RR data seems noisy, indicating that the time between heartbeats is not well fixed, a feature called heart rate variability (HRV). The spectrogram shows that HRV is not purely random, but has a cyclic component that is stronger during quiet sleep.

Note the long periods of hypoxia (<<95%) when the alarm was off.

Alarm off



Alarm on


Time (minutes)
SpO2 (%)
PR/min
RR (s)
RRcore (s)
RRCore
Time
Normalized freq.