Remote Sensor Project

S.Matoike

2024 年 4 月 21 日

目次

1　リモートマシン

1.1　リモート機器、センサー

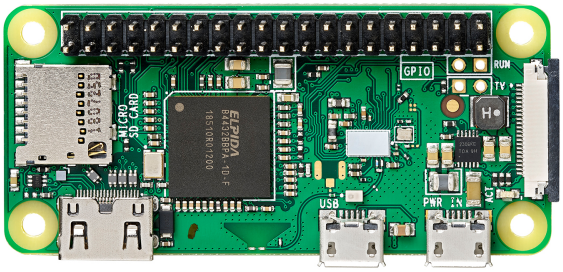

図 1　Raspberry Pi Zero WH

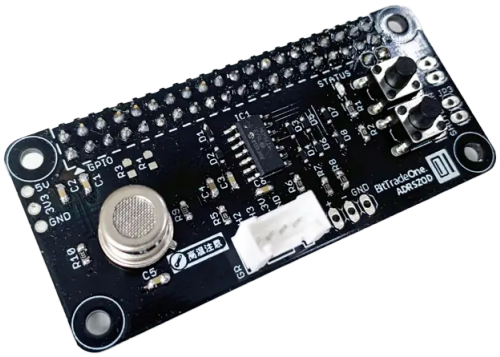

図 2　ADRSZOD:TP401A

1.2　リモート処理の概要

- Raspberrin Pi Zero WH 上に用意した shell（sensor.sh）だが、この shell は別の RaspberryPi マシンから ssh によって起動される
- i2C で接続された、臭気センサからの出力（電圧値）を、Python のプログラム（adrszOD.py）によって読み取ることができる
- adrszOD.py は、ch1 から ch4 までの値を出力しているので、その出力を awk にパイプして、1 つ目の ch1 の値だけを出力する様にしている
- 測定の日付を date コマンドで作成しているが、これは後ほど Python の matplotlib に合わせて、別途の整形処理が必要になることのない様に、書式を（年月日と時分秒の間は、半角カンマ＋半角スペース 1 個）の格好になるように整形している
- 日付の出力と、臭気センサの値が 1 行になる様にするため、paste コマンドを使っている
- paste の 2 つのオペランド、<(date +....)　と、<(python3 adrszOD.py...) の各のコマンドを、<(any command) の形に囲むことによって、2 つのファイルの様に扱う

ことができる（プロセス置換[*1]）

/home/mat/Documents/sensor.sh

```
#!/usr/bin/bash
dir='/home/mat/Documents'
paste <(date +%Y-%m-%d,\ %H:%M:%S) \
<(python3 "${dir}"/adrszOD.py | awk '{print $1}')
```

sensor.sh の出力例

```
mat@raspberrypizero:~/Documents $ ./sensor.sh
2024-04-17, 02:40:55 ch1:0.79827
mat@raspberrypizero:~/Documents $
```

[*1] 「Efficient Linux コマンドライン」オライリー・ジャパン の p.156 に、プロセス置換の解説がある

/home/mat/Documents/adrszOD.py

```
#!/usr/bin/env python
import smbus
import time

i2c = smbus.SMBus(1)
addr=0x68
Vref=2.048

def swap16(x):
  return (((x << 8) & 0xFF00) | ((x >> 8) & 0x00FF))

def sign16(x):
  return ( -(x & 0b1000000000000000) | (x & 0b0111111111111111) )

def readval(x):
  i2c.write_byte(addr, x) #16bit
  time.sleep(0.2)
  data = i2c.read_word_data(addr,0x00)
  raw = swap16(int(hex(data),16))
  raw_s = sign16(int(hex(raw),16))
  volts = round((Vref * raw_s / 32767),5)
  return volts

if __name__=="__main__":
  volts1 = readval( 0b10011000 )
  volts2 = readval( 0b10111000 )
  volts3 = readval( 0b11011000 )
  volts4 = readval( 0b11111000 )
  out_msg = 'ch1:' + str(volts1) + ' ,ch2:' + str(volts2)+\
    ' ,ch3:' + str(volts3)+ ' ,ch4:' + str(volts4)
  print(out_msg)
```

2　ローカルマシン

2.1　ローカル機器

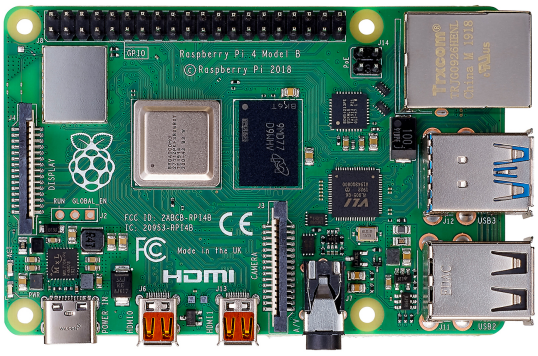

図 3　Raspberry Pi

2.2　cron による時刻指定処理

cron の仕組みを使って、3 つの時刻指定の処理を登録した

(1) 5 分ごとに、リモートセンサの値を読み取る処理、rsensor.sh を起動する
(2) 毎日 23 時 56 分になったら、その日 1 日分の測定データを退避する処理、dayly.sh を起動する
(3) 毎週日曜日の朝 0 時 1 分になったら、先週 1 週間の（先週の日曜日の 0 時 0 分から昨日の土曜日の 23 時 55 分まで 5 分間隔で測定した）測定データをまとめる処理、weekly.sh を起動する

crontab -e

```
MAILTO=""
*/5 * * * * sh /home/mat/Documents/rsensor.sh
56 23 * * * sh /home/mat/Documents/dayly.sh
1 0 * * Sun sh /home/mat/Documents/weekly.sh
```

ここで、MAILTO="" としているのは、「指定時刻に cron が処理を起動したよ」というメールを送信するために smtp を起動しようとして失敗し、指定の処理が起動されないと

いう現象に対する対応である[*2]

```
+---------------- minute (0 - 59)
|  +------------- hour (0 - 23)
|  |  +---------- day of month (1 - 31)
|  |  |  +------- month (1 - 12)
|  |  |  |  +---- day of week (0 - 6) (Sunday=0 or 7)
|  |  |  |  |
*  *  *  *  *  command to be executed
```

[*2] postfix をインストールせよという説もあるが、特にメールの知らせが不要ならこのやり方が最も簡易な解決方法になる

2.3　リモートセンサの処理をローカルから起動

リモート（192.168.3.27）の Raspberry Pi Zero WH に接続されている臭気センサの値を読みとる shell（sensor.sh）を、ローカル（192.168.3.21）のマシン Raspberry Pi から ssh を使って起動するための shell（rsensor.sh）であり、cron によって 5 分ごとに起動される

この時 ssh から、リモートマシン（Pi Zero）のユーザ（今の環境では mat）のパスワードを要求されるが、人手によるインタラクティブな入力操作を想定していないので、予め sshpass を導入しておいて、それによってパスワードを自動応答させる様にしている

ssh でリモートの shell を実行した結果を受け取り、ローカルの w.txt に追記し蓄積していっている

/home/mat/Documents/rsensor.sh

```
#!/usr/bin/bash
### sudo apt install -y sshpass ###
dir='/home/mat/Documents'
addr='192.168.3.27'
pwd='mypassword'
sshpass -p "${pwd}" ssh mat@"${addr}" "${dir}"/sensor.sh \
>> "${dir}"/w.txt
```

w.txt の様子は、tail -f w.txt などとしてみると、5 分ごとに起動された shell（rsensor.sh）によって、測定結果が蓄積されていく様子がわかる（このファイルは、1 日の終わりの処理で 2024-04-16.txt の様に、日付をつけたファイル名に変えて保存している）

w.txt -> 2024-04-16.txt

```
2024-04-16, 00:35:04 ch1:1.0896
2024-04-16, 00:40:04 ch1:1.09016
2024-04-16, 00:45:04 ch1:1.09322
.... ....
2024-04-16, 23:45:04 ch1:0.9059
2024-04-16, 23:50:04 ch1:0.99653
2024-04-16, 23:55:04 ch1:0.98809
```

2.4 日毎の処理

毎日の 23 時 55 分に、その日の最後の測定が終わる事を前提として、cron には 23 時 56 分になったら次の shell（dayly.sh）を起動する様にしている

現在時刻が 23 時 55 分を超えていることを確認した上で、次のことを行っている

- 蓄積された 1 日分の測定データ w.txt を、bkup.txt という名前のファイルにコピーしている
- w.txt を、data フォルダ以下に、「今日の日付.txt」 という名前に変更して保存している
- 空のファイル w.txt を次の日の測定データの蓄積場所のために用意している（touch）
- Python の仮想環境 venv11 に入って、daily.py を起動している（第 1 引数に data フォルダ以下に保存した測定データファイルの名前＝年月日を指定している）
- Python の仮想環境 venv11 を終えている

/home/mat/Documents/dayly.sh

```
#!/usr/bin/bash
dir='/home/mat/Documents'
dates=`date +%Y-%m-%d`
hour=`date +%H`
minu=`date +%M`
if [ "${hour}" -ge 23 ]; then
  if [ "${minu}" -gt 55 ]; then
    cp "${dir}"/w.txt "${dir}"/bkup.txt
    mv "${dir}"/w.txt "${dir}"/data/"${dates}".txt
    touch "${dir}"/w.txt
    source "${dir}"/venv11/bin/activate
      "${dir}"/venv11/bin/python3 "${dir}"/dayly.py "${dates}"
    deactivate
  fi
fi
```

この Python のプログラムを実行した結果として、次の様な 1 日分のデータの変化をグラフ化したものを、figs フォルダ以下に 「今日の日付.png」 という名称で保存している

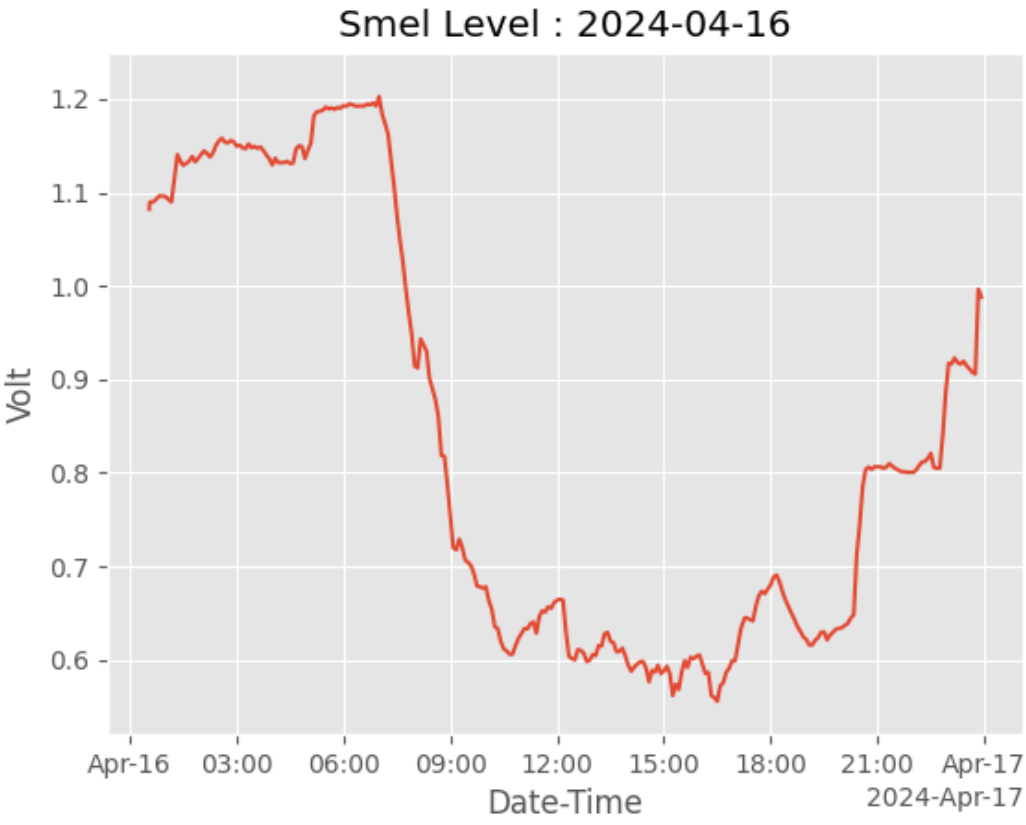

図 4　dayly.py の出力（4 月 16 日の 0 時から 24 時まで）

2.5　グラフ描画のための関数（myplot.py）

dayly.py と weekly.py で共通に呼び出している graph_plot() 関数を定義している

横軸に日付と時刻（データは 5 分毎）を置くため、少し工夫が必要になる

ポイントは、matplotlib が理解する日付の様式で書いたものを使うと、matplotlib の dates に、locator と formatter を自動生成させて、それ以下の全てを任せきることができる[*3]

[*3] https://matplotlib.org/stable/api/dates_api.html の記事が参考になる

/home/mat/Documents/myplot.py

```
import numpy as np, pandas as pd
from matplotlib import pyplot as plt, style, dates as mdates

def graph_plot(list0, dirstr, datestr, weekly=False):
  df = pd.DataFrame(list0, columns=["DateVal", "Value"])
  matplotlib.style.use('ggplot')
  fig, ax = plt.subplots()
  x = df.loc[:,"DateVal"].values
  y = df.loc[:,"Value"].values
  locator = mdates.AutoDateLocator()
  formatter = mdates.ConciseDateFormatter(locator)
  ax.xaxis.set_major_locator(locator)
  ax.xaxis.set_major_formatter(formatter)
  ymax = np.array([y.max(), 1.25]).max()
  ymin = np.array([y.min(), 0.52]).min()
  ax.set_ylim([ymin, ymax])
  ax.set_xlabel('Date-Time')
  ax.set_ylabel('Volt')
  if weekly:
    titlestr = "Smel Level : " + datestr + "(Weekly)"
    figpath = dirstr + "/figs/" + "Weekly_" + datestr
  else:
    titlestr = "Smel Level : " + datestr
    figpath = dirstr + "/figs/" + datestr
  ax.set_title(titlestr)
  ax.plot(x,y)
  fig.savefig(figpath + '.png')
  plt.show()
```

2.6 日毎処理のプログラム

- myplot.py から、graph_plot() 関数を import している
- 第 1 引数に指定があったら、そこで指定された日付のデータを元に、グラフを描画できる様にしている
- 指定のデータファイルを開いて、最初の 20 文字までは、予め matplotlib の dates に対応した様式の日付文字列にしているので、それを Unix の日付けに変換している
- 25 文字目以降は、測定した電圧の値だが、改行文字が付いていたりするので、.strip() によってホワイトスペースを除去して、実数に変換している
- 日付と電圧のリストを作って、それを graph_plot() 関数に渡して作図させている

/home/mat/Documents/dayly.py

```
import sys
from datetime import datetime as dt
import matplotlib.dates as mdates
from myplot import graph_plot

dirstr = '/home/mat/Documents'
today = dt.now().strftime('%Y-%m-%d')
if len(sys.argv) > 1:
  today = sys.argv[1]

list0=[]
with open(dirstr + "/data/" + today + ".txt") as f:
  for line in f:
    datestr = line[:20].strip()
    datev = mdates.datestr2num(datestr)
    value = float(line[25:].strip())
    list0.append([datev, value])

graph_plot(list0, dirstr, today, False)
```

2.7 週毎の処理

この処理は cron によって、毎週日曜日の朝、0 時 1 分に起動される

この shell（weekly.sh）では、0 時 0 分より後の時刻ならば仮想環境 venv11 に移って、Python のプログラム（weekly.py）を起動し、終わったら仮想環境鵜から抜けている

Python のプログラムは、第 1 引数に日付を受け取ることができる様にしている

/home/mat/Documents/weekly.sh

```
#!/usr/bin/bash
dir='/home/mat/Documents'
dates=`date +%Y-%m-%d`
hour=`date +%H`
minu=`date +%M`
if [ "${hour}" -ge 0 ]; then
  if [ "${minu}" -gt 0 ]; then
    source "${dir}"/venv11/bin/activate
      "${dir}"/venv11/bin/python3 "${dir}"/weekly.py "${dates}"
    deactivate
  fi
fi
```

2.8 週毎処理のプログラム

- myplot.py から graph_plot() 関数を import している
- 第 1 引数で処理対象にする最終日付を受け取ることができる様にしている
- 指定された処理対象の最終日付から、7 日前の日付を求めて、7 日間の日付のリスト datelist を作っている
- 7 日間の日付リスト datelist から 1 日ずつ取り出しては、そのファイル名のデータを data フォルダから読み出し、日付の変換と電圧の実数への変換を行い、それらのデータのリストを list0 に追記していっている
- もし、指定した日付のデータファイルがなかった場合は、例外処理で受けて黙って次の日付のファイルのデータ取得に移る様にしている
- graph_plot() 関数に list0 を渡してグラフの作図をさせている

/home/mat/Documents/weekly.py

```
from datetime import datetime as dt, timedelta
import sys, matplotlib.dates as mdates
from myplot import graph_plot

dirstr = '/home/mat/Documents'
s_format = '%Y-%m-%d'
today = dt.now()
if len(sys.argv) > 1:
  today = dt.strptime(sys.argv[1], s_format)
startstr = (today - timedelta(days=7)).strftime(s_format)

datelist = []
for i in range(7):
  day = today - timedelta(days=(7-i))
  datelist.append(day.strftime(s_format))

list0=[]
for fname in datelist:
  try:
    with open(dirstr + "/data/" + fname + ".txt") as f:
      for line in f:
        datestr = line[:20].strip()
        datev = mdates.datestr2num(datestr)
        value = float(line[25:].strip())
        list0.append([datev, value])
  except FileNotFoundError:
    continue

graph_plot(list0, dirstr, startstr, True)
```

以下は、週毎処理が出力する画像である。4 月 21 日（日曜日）の 0 時 1 分に cron によって起動され実行されたものだが、フォルダ data の中には、測定を始めた 4 月 16 日

（火曜日）から、4 月 20 日（土曜日）までの 5 日分のファイルがあるだけである。本来なら 1 週間前の日曜日（4 月 14 日）のデータから週毎処理の対象となるところだが、プログラムの中では FileNotFoundError の例外を捕捉して、とにかく continue によって処理を継続させているため、何事もなかったかの様に、データファイルの所在する火曜日以降がグラフに作図されている

また、日毎処理のプログラム dayly.py と全く同じグラフ作図の関数（myplot.py の中の graph_plot() 関数）を使っているのだが、横軸の設定が matplotlib の dates（mdates としてプログラム中では使っている）によって、自動的に調整されているのが分かる（便利）

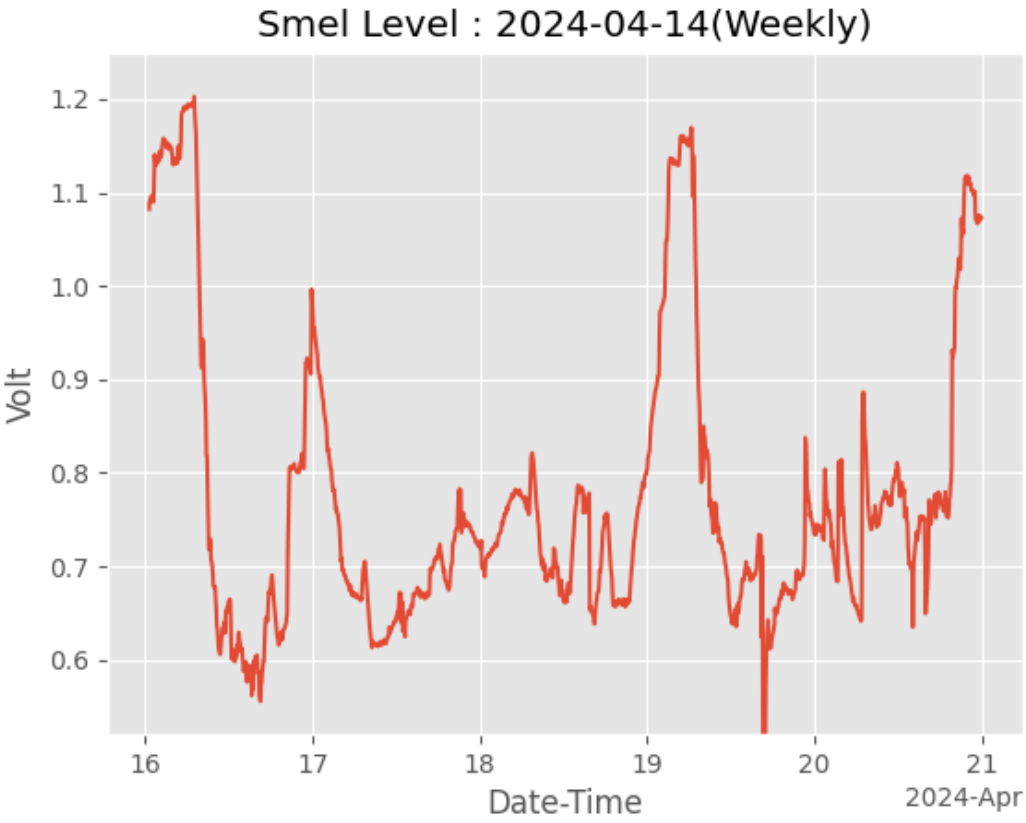

図 5　weekly.py の出力

2.9 測定データグラフの実時間描画

- RTPlot クラスに必要な関数類をまとめた
- ポイントとなるのは、 plt.show() を使わず、plt.pause(1.0) を使うこと、tail -f をプログラム上で実現することの 2 点である
- self.x と self.y のリストに順にデータを追記していき、新たなデータが w.txt に追加されたタイミングで plot.pause(1.0) としている（引数の値を 0.1 や 0.01 にすると Raspberry Pi では全て描画しきれないので注意）
- tail_f() 関数は、コマンド tail -f w.txt を Python のプログラムで実現したもの（指定したファイルの最終行が追記されるまで continue で待っている）である

```
import sys, time, numpy as np, pandas as pd
from matplotlib import style, pyplot as plt, dates as mdates
from datetime import datetime as dt, timedelta

class RTPlot:
    def dateproc(self):
        #s_format = '%Y-%m-%d, %H:%M:%S'
        s_format0 = '%Y-%m-%d, 00:00:00'
        today = dt.now()
        tomor = today + timedelta(days=1)
        self.todaystr = today.strftime(s_format0)
        self.tomorstr = tomor.strftime(s_format0)

    def readdata(self, fname):
        try:
            with open(fname, 'r') as f:
                for line in f:
                    self.chop(line)
        except FileNotFoundError:
            print(f'File{fname} not found')
```

```
    def __init__(self, fname):
        self.fname = fname
        self.x = []
        self.y = []
        self.ims = []
        self.dateproc()
        self.readdata(fname)
        self.fig, self.ax = plt.subplots()
        locator = mdates.AutoDateLocator()
        formatter = mdates.ConciseDateFormatter(locator)
        self.ax.xaxis.set_major_locator(locator)
        self.ax.xaxis.set_major_formatter(formatter)
        xmin = mdates.datestr2num(self.todaystr)
        xmax = mdates.datestr2num(self.tomorstr)
        self.ax.set_xlim([xmin, xmax])
        ymin, ymax = 0.52, 1.25
        self.ax.set_ylim([ymin, ymax])
        self.ax.set_title("Smel Level : " + self.todaystr[:10] +\
                          " :Real time plot")
        self.ax.set_xlabel('Date-Time')
        self.ax.set_ylabel('Volt')
        matplotlib.style.use('ggplot')
        im, = self.ax.plot(self.x, self.y)
        self.ims.append(im)

    def chop(self, line):
        datestr = line[:20].strip()
        datev = mdates.datestr2num(datestr)
        value = float(line[25:].strip())
        self.x.append(datev)
        self.y.append(value)

    def update(self):
```

```
        if len(self.ims) > 0:
            im = self.ims.pop()
            im.remove()
        im, = self.ax.plot(self.x, self.y)
        self.ims.append(im)

    def tail_f(self):
        try:
            with open(self.fname, 'r') as f:
                f.seek(0, 2)
                while True:
                    line = f.readline()
                    if not line:
                        time.sleep(1)
                        continue
                    return line.strip()
        except FileNotFoundError:
            print(f'File{self.fname} not found')
        return ""

if __name__=="__main__":
    dirstr = '/home/mat/Documents'
    fname = dirstr + '/w.txt'
    if len(sys.argv) > 1:
        fname = dirstr + '/' + sys.argv[1]
    rtp = RTPlot(fname)
    tomorrow = dt.now() + timedelta(days=1)
    while tomorrow - dt.now() > timedelta(minutes=5):
        plt.pause(1)
        line = rtp.tail_f()
        print("[info.log]", line)
        rtp.chop(line)
        rtp.update()
```

```
    rtp.fig.savefig(dirstr+'/figs/R_'+rtp.todaystr+'.png')
```

main での反復条件だが、当初 「dt.now() < tomorrow の間繰り返せ」としていたが、この条件だと tail_f() 関数から戻れなくなる（23 時 55 分に w.txt が更新された後は、tail_f() の中で continue がひたすら繰り返されている）ので、最後に tail_f() から戻ったタイミングを捉えて while を break しなければならない。w.txt の最後の更新が 23 時 55 分を過ぎた数秒後辺りだとすれば、それが tail_f() がら戻る最後のタイミングであり、while を break する時なので、反復を続ける条件を「tomorrow - dt.now() > timedelta(minutes=5)」とした。（もし、minutes=4 などとしてしまったら、最終回になるべき呼び出しから戻った後には、 やおら tail_f() の中に飛んでしまい、戻れなくなっているかもしれない）

実時間処理の画面は次の様になる

この画面を見ると概ね予定通りの出力になっているのだが、style.use('ggplot') の指定が無視されているのは気になる（なぜなのか、現段階で「私には」分かっていない）

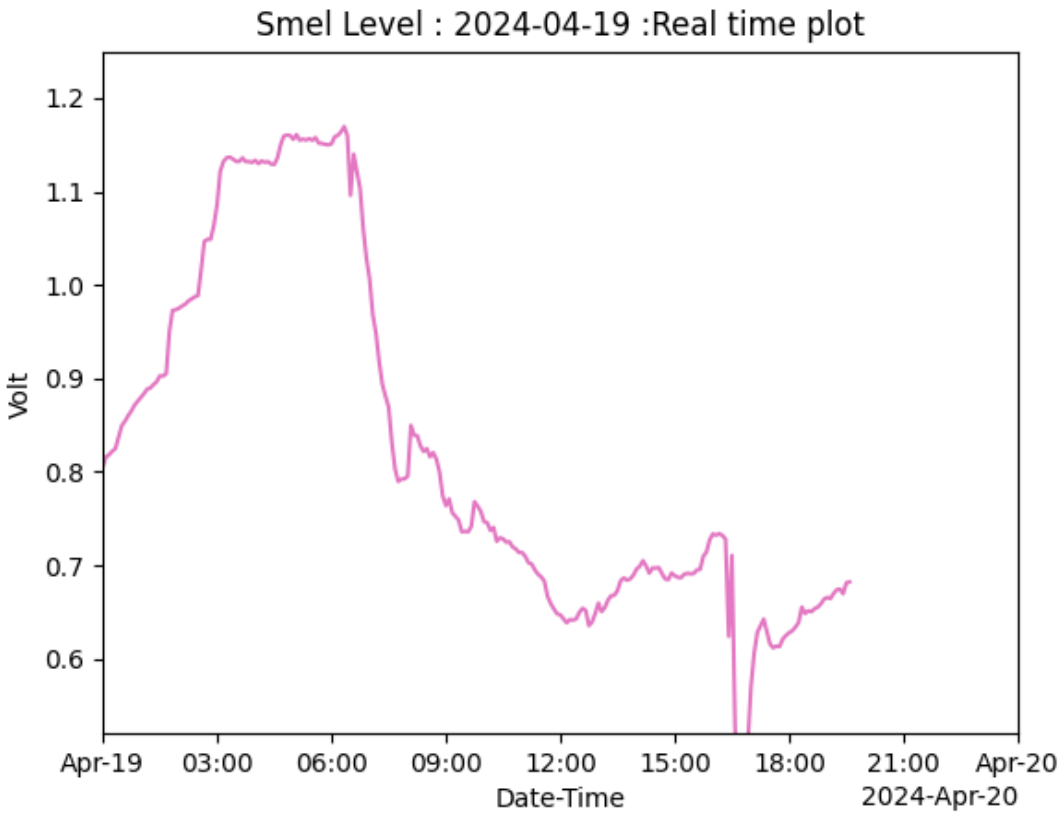

図 6　実時間出力の例（途中経過：4 月 19 日の 20 時頃）

3　参考文献

- 臭気センサ拡張基板とそのサンプルプログラム
  https://github.com/bit-trade-one/RasPi-Zero-One-Series
- matplotlib が認識する日付の書式
  https://matplotlib.org/stable/api/dates_api.html
- プロセス置換
  Daniel J.Barrett 著、オライリー・ジャパン「Efficient Linux コマンドライン」初版第 1 刷、p.157 のコラム記事