

# Remote Sensor Project

S.Matoike

2024年5月5日

## 目次

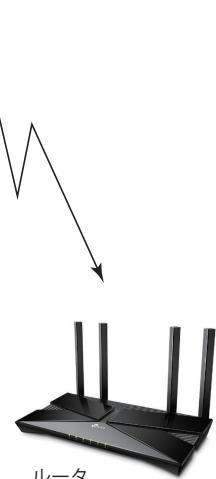
1	全体の構成	2
2	リモートマシン	3
2.1	リモート機器、センサー	3
2.2	リモート処理の概要	4
3	ローカルマシン	6
3.1	ローカル機器	6
3.2	cronによる時刻指定の処理	7
3.3	リモートセンサの処理をローカルマシンから起動する	8
3.4	日毎の処理	9
3.5	日毎処理のプログラム	10
3.6	グラフ描画のための関数	11
3.7	週毎の処理	13
3.8	週毎処理のプログラム	13
3.9	測定データのグラフを実時間で描画する	16
4	新たなシステムへの移行手順	21
4.1	ローカルマシンの移行	21
4.2	リモートマシンの移行	23
5	参考文献	24

# 1 全体の構成

[ 192.168.3.27 ]



リモートマシン  
Raspberry Pi Zero WH  
ゼロ・ワン臭気センサ



[ 192.168.3.21 ]



ローカルマシン  
Raspberry Pi 4 or 5  
US キーボード、マウス (USB)  
モニタ (HDMI)

図 1 システム全体の構成

- 5 分毎にリモートマシン上の臭気センサの測定データをローカルマシンに読み取る
- 読み取った 5 分毎のデータは、追記、蓄積していくと同時に、実時間の臭気変化としてグラフを描画更新する（ローカルマシン上）
- 23 時 56 分に、日毎の処理を起動し、1 日分の蓄積データをグラフ化すると共に、蓄積した 1 日分のデータは別フォルダに退避する（ローカルマシン上）
- 日曜日の 0 時 1 分に、週毎の処理を起動し、その日から過去 1 週間分の蓄積データを読んでグラフ化する（ローカルマシン上）

## 2 リモートマシン

### 2.1 リモート機器、センサー

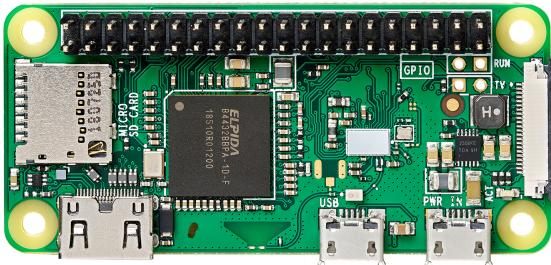


図 2 Raspberry Pi Zero WH



図 3 ADRSZOD:TP401A

#### 臭気センサ TP-401A

- 低電力で動作可能な高感度の臭気センサ
- コンパクトな Raspberry Pi Zero と重ねて使用可能な pHAT サイズ
- アンモニア、水素、アルコール、一酸化炭素、メタンなど揮発性気体、タバコ、木材燃焼発生した煙など様々な気体に対応
- 連続通電を許容するので、様々な所での継続的なチェックが可能（動作時にはセンサ部が非常に高温となるため、取り扱いには注意）
- 動作状況が見えるステータス LED により、駆動状況の確認が容易

#### 運用方針

- 基本的な運用形態は、モニタ、キーボード等を接続せず電源のみの接続で運用する
- このマシン上では、数行の shell を登録するだけなので、基本的には他の PC のコンソールから ssh で login して利用する（この例では、ユーザは mat）

```
ssh mat@192.168.3.27
```

- VNC を有効にして他の PC からアクセスすれば、デスクトップも利用できる

## 2.2 リモート処理の概要

- Raspberrin Pi Zero WH 上に用意したこの shell (sensor.sh) 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の出力 (電圧値) は、Python のプログラム (adrszOD.py) によって読み取る
- 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 の値だけを取り出す
- date コマンドで作成している測定日付は、後ほど Python の matplotlib の都合に合わせて、新たにデータの整形処理が必要にならない様に、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」の格好になる様に整形する
- 日付とセンサの出力する値が 1 行になる様にするため、paste コマンドを使う
- paste の 2 つのオペランド、<(date +....) と、<(python3 adrszOD.py...) の各々のコマンドは、<(any command) の形に囲むことにより、2 つのファイルの様に扱うことができる (プロセス置換<sup>\*1</sup>)

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

この shell を起動してみると、次の通り

sensor.sh の出力例

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

以下は、センサーを読むための Python プログラムのソース (臭気センサ拡張基板のメーカーが提供しているプログラム<sup>\*2</sup>を書き変えて使っている)

<sup>\*1</sup> 「Efficient Linux コマンドライン」 オライリー・ジャパン 初版第 1 刷 p.156 (プロセス置換の解説)

<sup>\*2</sup> <https://github.com/bit-trade-one/RasPi-Zero-One-Series>

```

/home/mat/Documents/adrszOD.py ━━━━━━━━

import smbus
import time

class I2C:
    i2c_bus = None
    def __init__(self, bus_number=1):
        cls = type(self)      # 旧スタイルなら、cls = self.__class__
        cls.i2c_bus = smbus.SMBus(bus_number)

class TP401A(I2C):
    def __init__(self, address=0x68, Vref=2.048) -> None:
        super().__init__(bus_number=1)
        self.address = address
        self.Vref = Vref

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

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

    def read_val(self, x):
        self.i2c_bus.write_byte(self.address, x) #16bit
        time.sleep(0.2)
        data = self.i2c_bus.read_word_data(self.address, 0x00)
        raw = self.swap16(int(hex(data),16))
        raw_s = self.sign16(int(hex(raw),16))
        volts = round((self.Vref * raw_s / 32767),5)
        return volts

```

クラス TP401A が、クラス I2C を継承する様にしたのは、新たな別の I2C 対応デバイスを、同じバス上に接続する場合を想定したもの

TP401A クラスは I2C クラスを継承しているので、TP401A のコンストラクタの中で I2C クラスのコンストラクタを明示的に実行し、I2C バスの番号を指定している

TP401A クラスの上位クラス I2C クラスのクラス変数 i2c\_bus へのアクセスは、「self. 変数名」の格好でアクセスすることによって、インスタンス変数→クラス変数→親クラスのクラス変数の順に探索され、I2C クラスのクラス変数にたどり着けることになる

この TP401A クラスを使う次の主処理は、sensor.sh の中から呼び出されている

```
/home/mat/Documents/procmain.py —————
```

```
#!/usr/bin/env python
from sarz0D import TP401A

if __name__ == "__main__":
    smel = TP401A()
    volts1 = smel.read_val( 0b10011000 )
    volts2 = smel.read_val( 0b10111000 )
    volts3 = smel.read_val( 0b11011000 )
    volts4 = smel.read_val( 0b11111000 )
    out_msg = f"ch1:{volts1} ,ch2:{volts2},ch3:{volts3},ch4:{volts4}"
    print(out_msg)
```

### 3 口一カルマシン

#### 3.1 口一カル機器

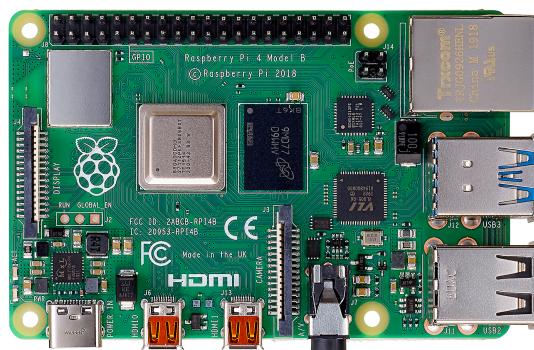


図 4 Raspberry Pi

### 3.2 cron による時刻指定の処理

cron の仕組みを使うことにする（3つの時刻指定の処理を登録している）

- (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、daily.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/daily.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<sup>3</sup>

```
+----- 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
```

<sup>3</sup> postfix や sendmail など、SMTP の何某か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

### 3.3 リモートセンサの処理をローカルマシンから起動する

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る

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

### 3.4 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 (daily.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）
- プログラムの実行を終えたら、Python の仮想環境 venv11 から抜けている

```
/home/mat/Documents/daily.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 "${dir}"/daily.py "${dates}"
        deactivate
    fi
fi
```

### 3.5 日毎処理のプログラム

- myplot.py から、graph\_plot() 関数を import している（この関数の詳細は後述）
- 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/daily.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)
```

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が得られる（これは `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）

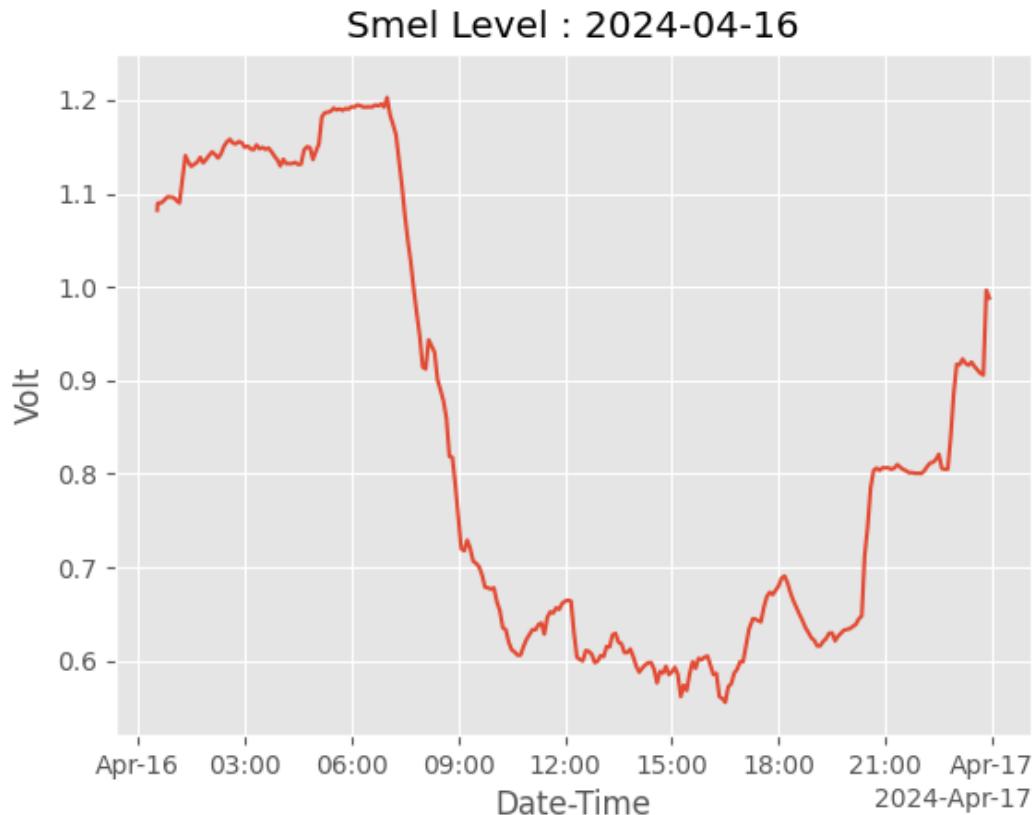


図 5 daily.py の出力 (4月 16 日の 0 時から 24 時まで)

### 3.6 グラフ描画のための関数

daily.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りができる。<sup>\*4</sup>

---

<sup>\*4</sup> [https://matplotlib.org/stable/api/dates\\_api.html](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)  
    ymin, ymax = y.min(), y.max()  
    step = (ymax - ymin) / 10.0  
    ymax = ((ymax // step) + 1) * step + step/2.0  
    ymin = (ymin // step) * step - step/2.0  
    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()
```

### 3.7 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
```

### 3.8 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れる様にしている
- 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 日ずつ取り出しても、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、例外処理で受けた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 output 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 daily.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）

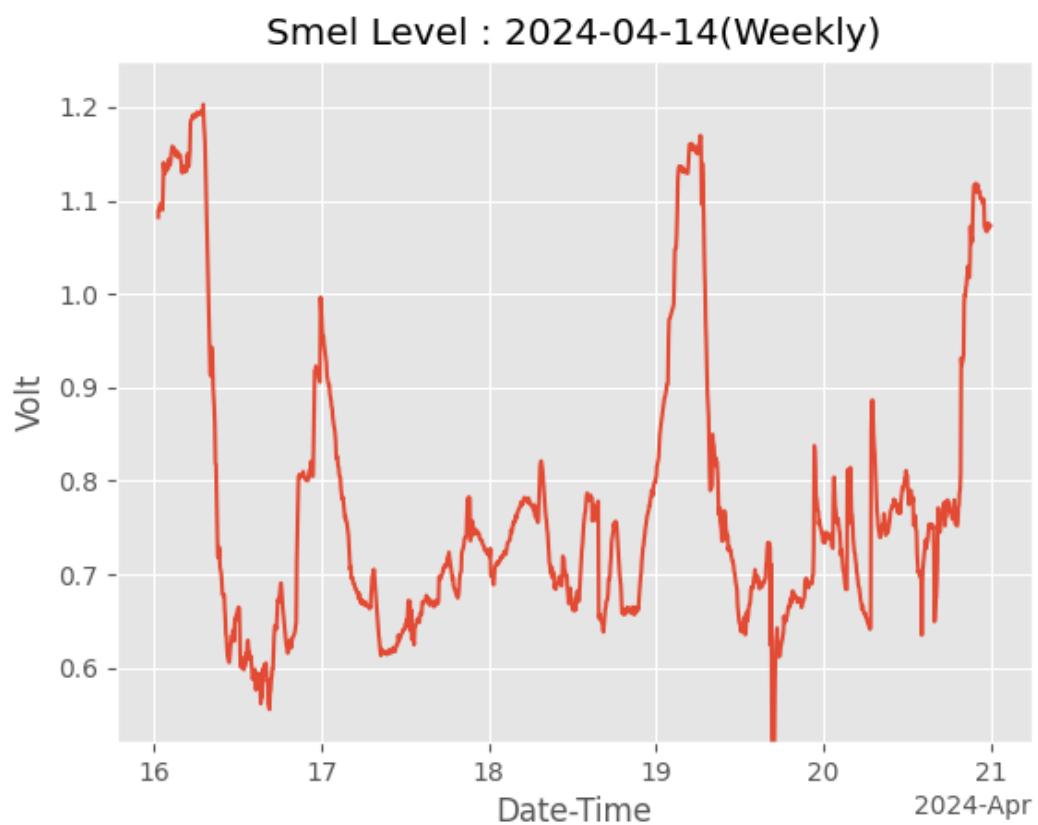


図 6 weekly.py の出力

### 3.9 測定データのグラフを実時間で描画する

- 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く、新たなデータが w.txt に追加されたタイミングで plt.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
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)
    matplotlib.style.use('ggplot')
    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])
    self.yrange()
    self.ax.set_xlabel('Date-Time')
    self.ax.set_ylabel('Volt')
    self.update()

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 statvalue(self):
    self.ymax = np.array(self.y).max()
    self.ymin = np.array(self.y).min()
    self.mean = np.array(self.y).mean()
    str1 = f"Max={self.ymax:.2f}, Min={self.ymin:.2f}, "

```

```

str2 = f"Mean={self.mean:.2f}"
self.ax.set_title("Smel Level : " + str1 + str2)

def tail_f(self):
    try:
        with open(self.fname, 'r') as f:
            f.seek(0, 2)
            enter = dt.now()
            while True:
                line = f.readline()
                if not line:
                    time.sleep(1)
                    if dt.now() - enter < timedelta(minutes=6):
                        continue
                return
            return line.strip()
    except FileNotFoundError:
        print(f'File{self.fname} not found')
    return

def yrange(self):
    self.statvalue()
    step = (self.ymax - self.ymin)/10.0
    v = 0
    while self.ymax > v:
        v += step
    ymax = v + step/2.0
    while self.ymin < v:
        v -= step
    ymin = v - step/2.0
    self.ax.set_ylim([ymin, ymax])

```

```

def update(self):
    self.yrange()
    if len(self ims) > 0:
        im = self ims.pop()
        im.remove()
        im, = self.ax.plot(self.x, self.y, color='red')
        self ims.append(im)

if __name__ == "__main__":
    dirstr = '/home/mat/Documents'
    fname = dirstr + '/w.txt'
    if len(sys.argv) > 1:
        fname = dirstr + '/' + sys.argv[1]
    rtp = RTPPlot(fname)
    tomorrow = dt.strptime(rtp.tomorstr, '%Y-%m-%d, 00:00:00')
    while dt.now() < tomorrow:
        plt.pause(1)
        line = rtp.tail_f()
        if line is not None:
            print("[info.log]", line)
            rtp.chop(line)
            rtp.update()
    rtp.fig.savefig(dirstr + '/figs/R_' + rtp.todaystr[:10] + '.png')

```

tail\_f() 関数の中では、6 分を超えて continue を繰り返している場合に None を返す事にしている（6 分待てば、その間には必ず 5 分間隔のイベントは入ってくると思う）ので、main の中の繰り返し処理 while の中にその事を（PEP8 が可読性の観点から推奨している書き方）is not None によって判定している

実時間処理の画面は次の様になる。

このプログラム rtplot.py が終了時に保存する画像は、実は毎日処理のプログラム daily.py が出力するものとほぼ同じ傾向を示しているはずだ。ただ rtplot.py の方は、新たな点をプロットするたびに y 軸の最大値と最小値を更新しているため、毎日の処理では作図の範囲外に出るデータも示せているし、また具体的な最大値、最小値、平均値の値もそ

の都度更新して表示している

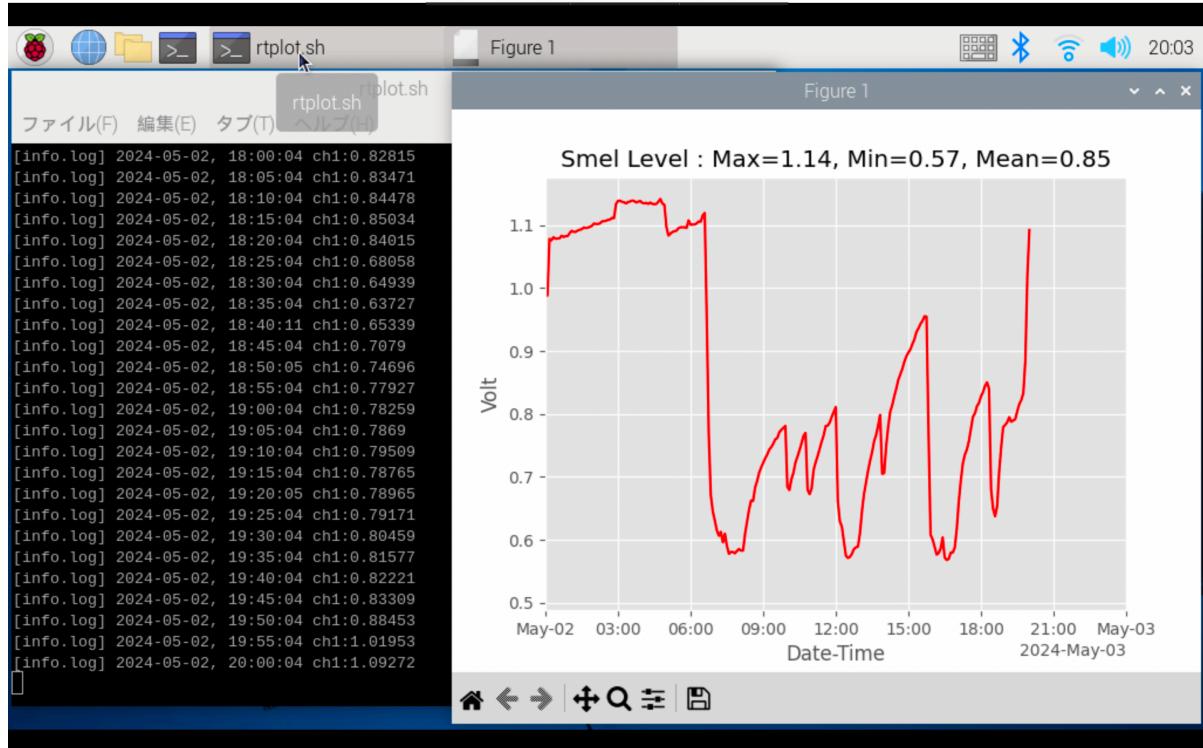


図 7 実時間出力（途中経過：5月2日の20時過ぎ）

この実時間プロットのプログラムを起動する shell を、実行可能属性を持たせてデスクトップに置いている（マウスのクリックで起動できる）

```
/home/mat/Desktop/rtplot.sh
```

```
#!/usr/bin/bash
dir='/home/mat/Documents'
cd "${dir}"
source "${dir}"/venv11/bin/activate
python3 "${dir}"/rtplot.py
deactivate
```

## 4 新たなシステムへの移行手順

### 4.1 ローカルマシンの移行

ここまで作成してきたものを、新たなマシン（Raspberry Pi）へ移行する際、必要な手続きについてまとめておく（新しいマシンの OS が導入され、ネットワークの設定を終えた後の手続きになる）

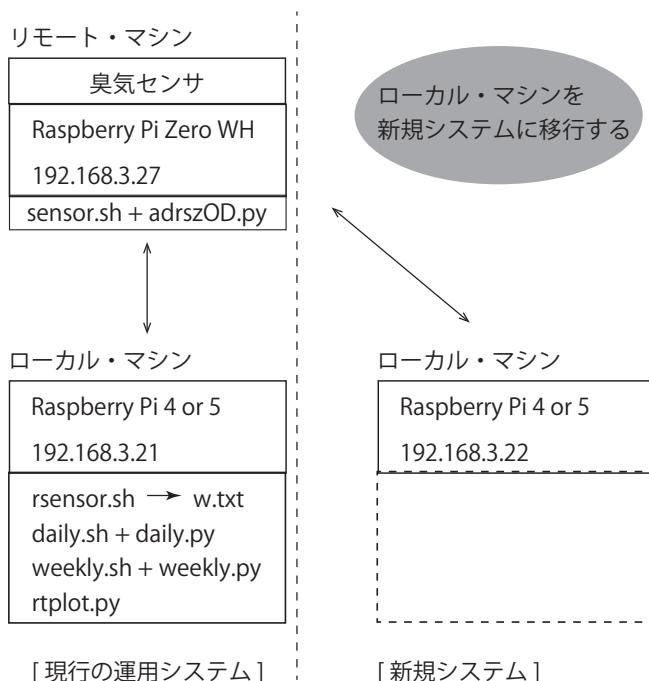


図 8 ローカルマシンの移行

以下の手続きはその都度全て、/home/mat/Documents の直下で実施するものとする

1. ssh を有効にする（Raspberry Pi の configuration で）
2. システムを最新の状態にする

```
sudo apt update  
sudo apt dist-upgrade
```

3. sshpass を導入する

```
sudo apt install -y sshpass
```

4. Python の仮想環境を /home/mat/Documents/venv11 につくる<sup>5</sup>

```
python3 -m venv venv11
source venv11/bin/activate
python3 -m pip install -U pip
pip install numpy==1.22.4
pip install scipy
pip install pandas==1.4.2
pip install matplotlib
pip install scikit-learn
pip list -o
deactivate
```

5. data フォルダと figs フォルダを移行する

```
scp -r mat@192.168.3.21:/home/mat/Documents/data ./
scp -r mat@192.168.3.21:/home/mat/Documents/figs ./
```

6. shell と Python のプログラムを移行する

```
scp mat@192.168.3.21:/home/mat/Documents/*.sh ./
scp mat@192.168.3.21:/home/mat/Documents/*.py ./
```

ここで、各 shell には実行可能属性がついている事を確認する ls -l \*.sh

もし付いていなかったら付ける chmod +x \*.sh

また、リモートマシンの IP アドレス、ユーザ名とそのパスワードが変わった場合は、

ここで rsensor.sh の当該箇所（露に記述している）を編集する

Python のプログラムに実行可能属性は不要である

7. cron に時刻指定手続きを登録する (crontab -e)

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

---

<sup>5</sup> 寺田学 他 3 名、翔泳社「Python によるあたらしいデータ分析の教科書」第 2 版、p.50

8. ssh でリモートマシンに接続を試みて、key fingerprint を登録させる

```
ssh mat@192.168.3.27
The authenticity of host '192.168.3.27' can't be established.
ED25519 key fingerprint is SHA256:.....
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
ssh mat@192.168.3.27 /home/mat/Documents/sensor.sh
```

9. txt ファイルを移行する

```
scp mat@192.168.3.21:/home/mat/Documents/*.txt ./
```

10. rsensor.sh が 5 分ごとに起動され、w.txt を更新している事を確認する

```
tail -f w.txt
```

11. 実時間モニタ用のプログラムを起動する

```
source venv11/bin/activate
python3 ./rtplot.py
```

## 4.2 リモートマシンの移行

新しいリモートマシンへの移行は次の通り

リモートマシンへの OS の導入とネットワークの設定を終えていること

1. configuration で、ssh と i2c を有効にする
2. 最新の状態に更新する

```
sudo apt update
sudo apt dist-upgrade
```

3. adrszOD.py と procmain.py を導入する
4. sensor.sh を導入して、実行可能属性をつける (chmod +x sensor.sh)
5. /home/mat/Documents/sensor.sh を実行して、現在日時とセンサの電圧、それぞれ適切な値が 1 行で出力される事を確認する

## 5 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](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
- Python の仮想環境  
寺田学 他著、翔泳社「Python によるあたらしいデータ分析の教科書」第2版、p.50