水密です

～“水中版”天気予報を目指して～

早稲田大学系属早稲田実業学校中等部高等部

坂　功弥・藤野克彬・押田育英・俵　秀成

Abstract

We developed the device to collect the data of the underwater environment. Nothing is to be found simply researching underwater, we developed this device to predict how the underwater environment will change in the future. This time, we programmed it to collect data of the underwater temperature with the inner sensor protected by the waterproof shell we made. We found this device is able to check the underwater temperatures.

1.　目的および先行研究

気象観測において, 陸上とは異なり, 環境が厳しい水中での観測を主目的にした機器はほとんど存在しない.

本実験では，水中ロボットを自作し，それを用いて水温の測定を行い水中の状態を観測することを目的とした.

昨年まで, 本校では海洋研究開発機構 JAMSTEC で行われた水中ロボットコンベンション in JAMSTEC に参加していた．その時の経験を基にして，今回の機材を開発した.

2.　観測に使用した器具

2.1.　本体の構造

本体は二つの部分で構成されている（図 1）. 左側に見えるものが, センサーや回路などの入った耐水殻で，防水処理を施してある．右側に見えるものがその耐水殻を支える枠となっていて，主に塩ビパイプで構成されている．実験時には，枠にこの耐水殻をはめて水中に入れるという構造になっている.

図 1　観測に使用した機器

2.2.　耐水殻

センサーや回路などが水に触れてしまうと，回路がショートしてしまい正常に動作しない. そこで, これらのものを耐水殻（240×130×60mm）に入れて防水仕様とした.

図 2 は耐水殻を上から見た図である．耐水殻には西川精機製作所製の耐圧殻を使用した．防水のため耐水殻中に回路を入れた後にティッシュを詰め，ゴムパッキンをはさんだ後に蓋をしてねじ止めをし，最後に周りをビニールテープで囲った．蓋の上に見える金属製のものは錘である．耐水殻の側面には 7 つの穴があり，これらに温度センサーのコードを通している. センサーを通していない穴にはダミーのコードを通している．この穴の防水には，特別なものを使用している（図 3）. それは，コードを穴に通した後に，手前側の六角形の蓋を時計回りにねじることで中に入っているゴムが圧迫されて防水となる仕

図 2　耐水殻

図 3　防水の工夫

組みになっている.

これらは中にコードが入っていないと防水効果を発揮できないため, 使わないところもダミーのコードを入れている. また, 図 4 に示すように, 前方の穴にはこの黒いジョイントを使わずに, 透明のアクリル板でふさいでいる. これは, 将来的にここにカメラを搭載するために視界を確保するためだ. 接着にはアロンアルファを使用した.

図 4　コードを固定するジョイントとカメラ用の穴

2.3.　枠

枠（250×240×160mm）は塩ビパイプを購入し製作した. もともとは全面に網を張っていたが, カメラを設置するときの視界の妨げになるので底面以外を撤去した.

図 5 は, 耐水殻を固定するのに使った枠である. 実験の際には青い網が下に来るように配置する. この枠の役割は三つある. 一つ目は耐水殻が川に流されずに水底にとどまっていられるように固定する役割である. 二つ目は耐水殻が外れた時に流されて行かないようにキャッチをする役割である. 三つ目は耐水殻を水から引き上げるときの手すりである. 後ろに見える白いものはロープで, 枠ごと流されないように地上から抑えられるようになっている. 黄色いものはチェーンであり, 川底の小石などに引っかかりやすくするためのものである.

図 5　耐水殻を固定する枠

3.　観測に使用した器具について

センサーは光センサー(NJL7502L)と温度センサー（LM35DZ）を，AD コンバーターは（MCP3208-CI/P）を使用した．しかしながら光量の定量的な測定は機体に固定されているセンサーの傾きに大きく左右されると考えられたため測定しなかった．

回路は図 6 のようになっている．

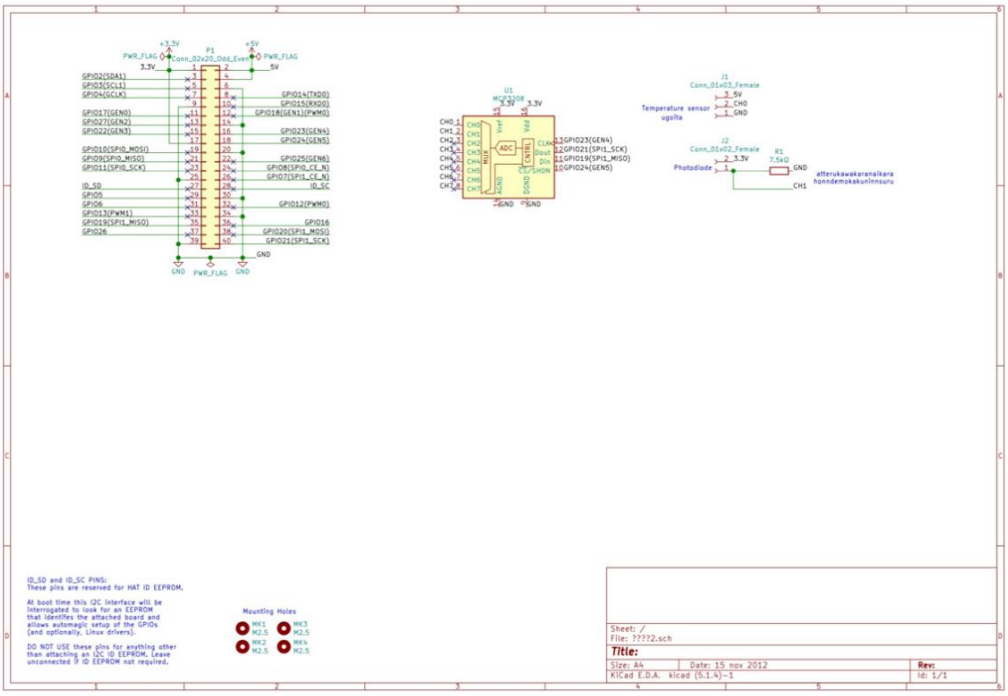


図 6　観測機器の回路図

給電はエレコムのモバイルバッテリー(DE-C10L-6400)から行なった．

Raspberry Pi Zero WH の Typical bare-board active current consumption（典型的なベアボードの有効消費電流）は 150mA で，モバイルバッテリーの容量は 6400ｍAh であるので約 42 時間稼働することになる．ただしこの数値は Raspberry Pi Zero WH に接続する機器によって大きく左右される．

ブレッドボード上で 0.1 μFのコンデンサを用いて電源信号のノイズを取り除く必要があるか検証したが，変化が見られなかったためコンデンサは省いた．

AD コンバーターの MCP3208 は 12bit，通信方式は SPI である．

温度センサーは最大 10mV の誤差がある．そのためこの AD コンバーターで温度センサーのアナログ信号を充分な分解能（1.22mV）でデジタル信号へ変換することができる．

<センサーの工夫>

耐水殻から出ている温度センサーの防水のためにホットボンドを使用した．シーリング材を使用する予定であったが，ホットボンドでも防水ができるのではないかと考え代用した．観測において防水性に問題は見られなかった．

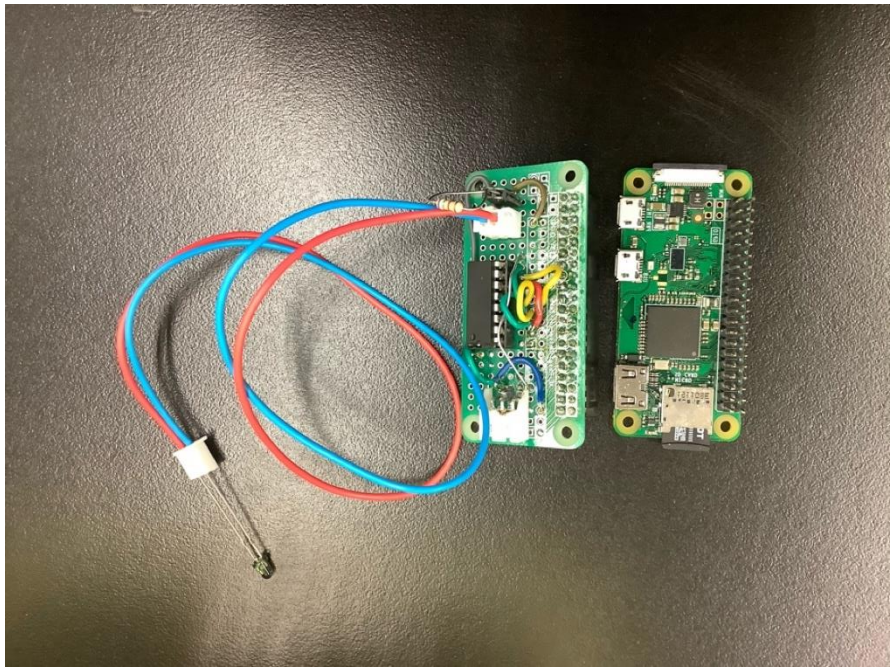


図 7　RaspberryPi zero WH（右）とセンサー基板（左）

4. ソフトウェア

4.1. 課題

IC 温度センサー（LM61CIZ）が接続されている AD コンバーター（MCP3208）の値を，SPI 通信をして RaspberryPi で定期的に読み取る．

AD コンバーターから読み取った電圧を，それぞれ適した単位に換算する．

以上の値を，まとめてテキストファイル[.txt]に記録する．（とともに，記録データを画面に表示する）

4.2. 目的

各種センサーの値を定期的に得て，まとめて記録する．このプログラムを作成することにより，観測データを一つにまとめることができ，観測の効率化につながる．

4.3. 装置および環境

今回用いた装置は表 1 の通りである．

表 1　使用装置

装置名	型番	使用用途
Raspberry Pi Zero WH	Raspberry Pi Zero WH	メイン処理・記録
AD コンバーター	MCP3208 2.7V 4-Channel/8-Channel 12-Bit A/D Converters with SPI Serial Interface	アナログ入力換算
IC 温度センサー	LM61CIZ	温度計測
フォトトランジスタ	NJL7502L	照度計測

また，今回の実行環境は表 2 の通りである

表 2　実行環境

CPU	1000MHZ シングルコア ARM1176JZ-F(ARMV6)
OS	RASPBIAN GNU/LINUX 10 (BUSTER)
メモリ	512 MB RAM
コンパイラ	GCC 8.3.0
PYTHON	PYTHON 3.8.6

4.4. 実行方法

・メインフォルダ直下にある[main.py]を実行することによって，プログラム開始.

・[main.py]内で指定されている観測間隔に沿って，観測・換算・記録・表示が繰り返される.

・[main.py]内で指定されている実行継続時間（プログラムを実行させておく時間の長さのこと）になると，プログラムが自動終了する.

・もし緊急で終了する場合は，Ctrl＋C で強制終了する．

4.5. 実行結果の例

表 3　実行結果の例

[1]	**$ python3 main.py**
[2]	start
[3]	datetime=2020-11-05 15:45:10.624622, temp=21.63525390625
[4]	datetime=2020-11-05 15:45:10.627967, lx=0.0Lux
[5]	++++++++++++++++++++++++++++++
[6]	datetime=2020-11-05 15:45:11.639079, temp=21.90527343749999
[7]	datetime=2020-11-05 15:45:11.641383, lx=0.0Lux
[8]	++++++++++++++++++++++++++++++
[9]	datetime=2020-11-05 15:45:12.649349, temp=20.13281249999999
[10]	datetime=2020-11-05 15:45:12.651549, lx=0.0Lux
[11]	++++++++++++++++++++++++++++++
[12]	datetime=2020-11-05 15:45:13.658976, temp=22.71093749999999
[13]	datetime=2020-11-05 15:45:13.661034, lx=0.2384765625Lux
[14]	++++++++++++++++++++++++++++++
[15]	datetime=2020-11-05 15:45:14.668988, temp=21.53564453124999
[16]	datetime=2020-11-05 15:45:14.671314, lx=0.0Lux
[17]	++++++++++++++++++++++++++++++
[18]	datetime=2020-11-05 15:45:15.688320, temp=22.21337890625
[19]	datetime=2020-11-05 15:45:15.690500, lx=0.2384765625Lux
[20]	++++++++++++++++++++++++++++++
[21]	**$■**

・1 行目で[main.py]を実行し，プログラム開始．

・2 行目の[start]は，プログラム開始を意味する．

・3～5 行目・6～8 行目・・・などの，“＋”で区切られている 3 行分を 1 サイクルとして，設定した実行継続時間になるまで，このサイクルが繰り返される．

・20 行目 6 サイクルでプログラムが終了し，21 行目でコンソールに戻っている．

また，DATA.txt を以下に示す

表 4　DATA.txt の例

[datetime=2020-11-05 15:45:10.624622,temp[°C]=21.63525390625]
[datetime=2020-11-05 15:45:10.627967,lx[Lux]=0.0]
[datetime=2020-11-05 15:45:11.639079,temp[°C]=21.90527343749999]
[datetime=2020-11-05 15:45:11.641383,lx[Lux]=0.0]
[datetime=2020-11-05 15:45:12.649349,temp[°C]=20.13281249999999]
[datetime=2020-11-05 15:45:12.651549,lx[Lux]=0.0]
[datetime=2020-11-05 15:45:13.658976,temp[°C]=22.71093749999999]
[datetime=2020-11-05 15:45:13.661034,lx[Lux]=0.2384765625]
[datetime=2020-11-05 15:45:14.668988,temp[°C]=21.53564453124999]
[datetime=2020-11-05 15:45:14.671314,lx[Lux]=0.0]
[datetime=2020-11-05 15:45:15.688320,temp[°C]=22.21337890625]
[datetime=2020-11-05 15:45:15.690500,lx[Lux]=0.2384765625]

このように，観測時刻と（換算された）センサーの値が記録される．

4.6.　理論

RaspberryPi や AD コンバーターのインターフェースと入出力については, 4 ページを参照.

・AD コンバーター（MCP3208）

読み取るチャンネル CHx（0 <= x <= 7）や通信方法は，AD コンバーターとの通信ビットのうち，通信パラメータの部分で設定できる．

通信パラメータは右記（表 5[i]）のように，

1(High) or 0(Low)の 4 桁の組み合わせで設定できる．

表 5　CONFIGURATION BITS FOR THE MCP3208

THE MCP3208

Control Bit Selections				Input Configuration	Channel Selection
Single /Diff	D2	D1	D0		
1	0	0	0	single-ended	CH0
1	0	0	1	single-ended	CH1
1	0	1	0	single-ended	CH2
1	0	1	1	single-ended	CH3
1	1	0	0	single-ended	CH4
1	1	0	1	single-ended	CH5
1	1	1	0	single-ended	CH6
1	1	1	1	single-ended	CH7
0	0	0	0	differential	CH0 = IN+ CH1 = IN-
0	0	0	1	differential	CH0 = IN- CH1 = IN+
0	0	1	0	differential	CH2 = IN+ CH3 = IN-
0	0	1	1	differential	CH2 = IN- CH3 = IN+
0	1	0	0	differential	CH4 = IN+ CH5 = IN-
0	1	0	1	differential	CH4 = IN- CH5 = IN+
0	1	1	0	differential	CH6 = IN+ CH7 = IN-
0	1	1	1	differential	CH6 = IN- CH7 = IN+

また，AD コンバーターとの通信内容詳細は，図 8[ii]に示す.

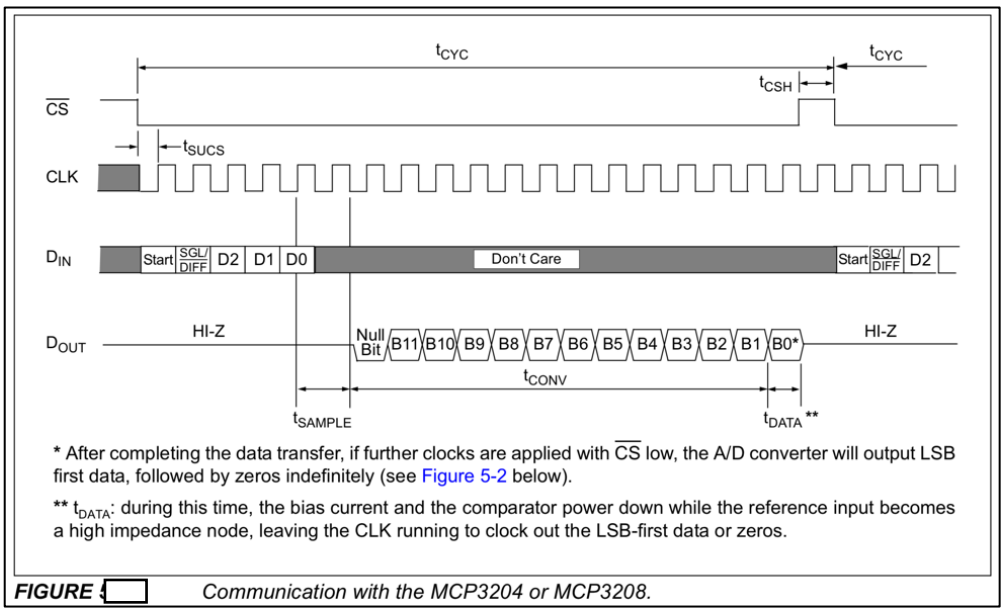


図 8　Communication with the MCP3204 or MCP3208.

このように，SPI 通信では 4 本の信号線（CS,CLK,Din,Dout）で通信する.

・CS（Chip Select）は，制御対象のデバイスを選択するための信号線だ.（Bring CS pin 0V[LOW]）実際に図 8 を見てみると，CS が Low の時に，他の信号線で通信していることが分かる.

・CLK(CLocK)は，通信するデバイス同士のタイミングを合わせるための同期用の信号線だ. 図 1 を見ると，CLK のクロックに合わせ，他の信号線で通信していることが分かる.

・Din はマスター（RaspberryPi）からスレーブ（AD コンバーター）方向にデータを送信するため，Dout はスレーブからマスター方向にデータを送信するための信号線である. FIGURE2 を見てみると，まず Din でマスターから通信パラメータなどを送信してから，Dout でスレーブから情報を読み取っていることが分かる.

これらを具体的な数値で表したものが以下の図 9[iii]である.

AD コンバーターから得られる値はあくまで電圧[V]のみのため，それをそれぞれ適した

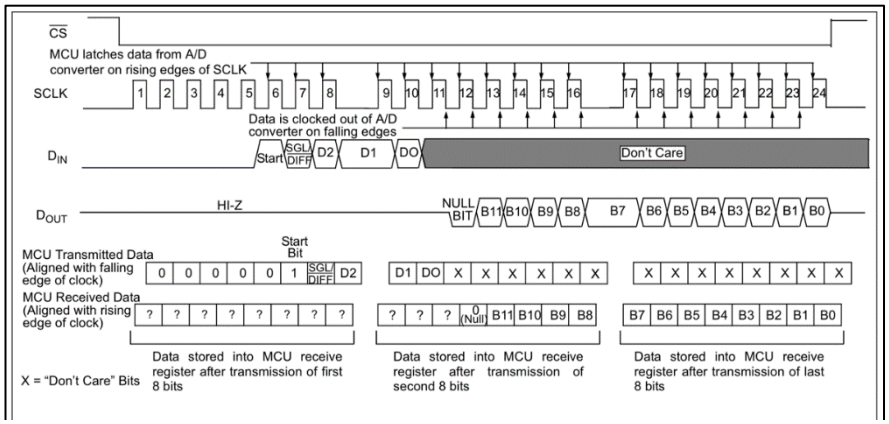


図 9　SPI Communication using 8-bit segments (Mode 0,0: SCLK idles low).

単位に換算する必要がある.

・IC 温度センサー（LM61CIZ）

このセンサーにおいて，電圧[V]から温度[℃]に換算するためには以下の式[iv]を使用する.

Vo=10mV/℃×T℃+600mV

よって整理して,

T(℃)=(Vo-0.6(V))/0.01(V)

V_O = 10 mV/°C × T°C + 600 mV

where

- T is the temperature in °C
- V_O is the LM61 output voltage

・フォトトランジスタ（NJL7502L）

このセンサーにおいて，電圧[V]から照度[Lux]に換算するためには以下の式を使用する.

$$\text{Illuminance[Lux]} = \left(\frac{E[V]}{R[k\Omega] \times 10^3} \times 10^6\right) \times 2.22$$

この式は，右記の図１０[v]を元に作成した.

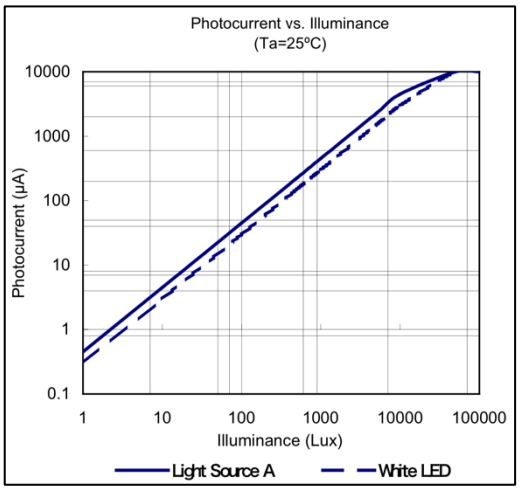


図 10　Photocurrent vs. Illuminance (Ta=25°C)

photocurrent[μA])

$$= Illuminance[Lux] \div 2.22$$

オームの法則　$I[A] = E[V]/R[\Omega]$　を代入して,

$$Illuminance[Lux] = \left(\frac{V[E]}{R[\Omega]} \times 10^6\right) \times 2.22$$

よって

$$\text{Illuminance[Lux]} = \left(\frac{E[V]}{R[k\Omega] \times 10^3} \times 10^6\right) \times 2.22$$

3.1.　プログラム構成

センサーの値を換算するための関数がとても多いため, メインの処理を行う[main.py]とは別に，独自モジュールとしてセンサー関連の関数は定義した.

図 11 にファイル構成図を表す.

```
Python/
  ├ OBSERVATION_DATA/
  ├ pythonmodules/
  |        ├ _init_.py
  |        ├ ADconverter_module.py
  |        └ sensor_module.py
  └ main.py
```

図 11 ファイル構成図

main.py

●定数・定義など

・sensor.lx_resistance

値：7.5

照度センサーに接続する抵抗値を代入

（“sensor.”で，[sensor_module.py]内の変数へアクセス）

・adc.Vref

値：3.3

AD コンバーターのリファンレス電圧（どの程度まで計測可能かを指定）

（”adc.”で，[ADconverter_module.py]内の変数へアクセス）

・spi_channel

値：0

SPI 接続機器の番号 CEx　x=0-1 (0-2 for the auxiliary SPI)

・baud

値：50000

通信速度 /32K-125M (values above 30M are unlikely to work)

・spi_flags

値：0

spi 接続時のオプション /　2 進数 22 ビットを 10 進数に変換して代入

・run_duration

値：30

実行継続時間 / (seconds)

●データファイル

・Python/OBSERVATION_DATA/<//プログラム実行時日時>/DATA.txt

観測したセンサーの値を記録するためのテキストファイル

ファイル内の構造は，11 ページのファイル構成図を参照。

init.py

独自モジュールだと認識させるためのファイル．中身は空白．

ADconverter_module.py

●定数・定義など

・Vref

値：None

リファンレス電圧（どの程度まで計測可能かを指定）

main.py からアクセスし，値を操作する．

●データ構造

・adch[] <class 'list'>：

A/D コンバーター[MCP3208]の読み取りたいチャンネル CHx（0 <= x <= 7）を設定する．

・data {} <class 'dict'>：

A/D コンバーターから取得した未加工値一覧

取得しなかったチャンネル CHx（0 <= x <= 7）は，<'None'>となる．

●関数の仕様

・get_MCP3208(adch)

表 6 def-get_MCP3208

説明	SPI 通信で AD コンバーター[MCP3208]から値を読むための関数.
参照する定数	hndl
引数	adch <class 'list'> : A/D コンバーター[MCP3208]の取得したいチャンネル(CHx)一覧
返り値	data <class 'dict'> : A/D コンバーターから得られた未加工値一覧 （取得しなかった CH は「None」）

sensor_module.py

●定数・定義など

・lx_resistance

値：None

照度センサーに接続する抵抗値

main.py からアクセスし，値を操作する.

●データ構造

・temp{} <class ‘dict’>

IC 温度センサーから取得した値を換算したものと時間をまとめた，最終的な結果をまとめる.

・lx{} <class ‘dict’>

フォトトランジスタから取得した値を換算したものと時間をまとめた，最終的な結果をまとめる.

●関数の仕様

・temp_LM61CIZ(volt)

表 7 def-temp_LM61CIZ

説明	IC 温度センサーLM61CIZ を接続した AD コンバーターからの値を，温度(℃)に変換する関数
引数	volt <class 'float'>：AD コンバーターからの電圧[V]
返り値	temp <class 'float'>：温度(℃)に変換した値

・lx_NJL7502L(volts)

表 8 def-lx_NJL7502L

説明	フォトトランジスタ NJL7502L を接続した AD コンバーターからの値を，照度(Lux)に変換する関数
参照する定数	lx_resistance
引数	volt <class 'float'>：AD コンバーターからの電圧[V]
返り値	lx <class 'float'>：照度(Lux)に変換した値

アルゴリズム

main.py

１．一般的なライブラリをインポート

２．自作モジュールをインポート

３．以下のプログラムを試みる（以下，初期設定を行う）

i. SPI 関係の変数・定数を宣言・代入

ii. デバイスオープン(Open SPI device on channel 0 in mode 0 at 50000 bits per second)

iii.　デバイスオープンできなかったら，プログラムを終了

４．3 で AttributeError が出たら，エラーメッセージを表示し，プログラムを終了

５．「start」と表示する．（特に意味はなし）

６．フォルダ[OBSERVATION_DATA/]を作成

７．カレントディレクトリを[OBSERVATION_DATA]に移動

８．現在日時を取得

９．「OBSERVATION_DATA」下に，記録フォルダ[<実行時刻名>/]を作成．その下に，写真用フォルダ[pictures/]を作成

１０．　カレントディレクトリを記録フォルダ[<実行時刻名>/]に移動

１１．　変数[DATA_txt]に[DATA.txt]ファイルを設定

１２．　観測データ記録用の[DATA.txt]テキストデータを作成

１３．　「DATA.txt」を開く（書き込み専用）

１４．　カレントディレクトリを元に戻す

１５．　実行継続時間を変数[run_duration]に代入し，設定する

１６．　現在時刻＋実行継続時間を計算し，終了予定時刻を求める

１７．　現在時刻が終了予定時刻よりも前である間：

i.　AD コンバーターの CH0,CH1 から情報を得る.（自作モジュールの関数を呼び出す.）
[Python/pythonmodules/ADconverter_module.py -> def get_MCP3208(adch)]

ii.　ｉから得られた情報を各関数（自作モジュール：以下）に渡し，それぞれ適した単位に換算する．それらのデータをリストにまとめ，テキストデータに記録.
また，画面にも表示する.
[Python/pythonmodules/sensor_module.py -> def temp_LM61CIZ(volt)]
[Python/pythonmodules/sensor_module.py -> lx_NJL7502L(volts)]

iii.　区切り線”+”を表示し，1 秒休み，17-iに戻って繰り返す.

１８．　SPI デバイスを閉じる

１９．　テキストを閉じる

２０．　プログラム終了

Python/pythonmodules/ sensor_module.py

・temp_LM61CIZ(volt)

1. 電圧[V]から温度[°C]に換算する
2. 換算した温度を返す

・temp_LM35DZ(volt)

1. 電圧[V]から照度[Lux]に換算する
2. 換算した照度を返す

Python/pythonmodules/ADconverter_module.py

1. 取得データ保存用のディクショナリ[data{}]を作成
2. 引数 adch[]で指定されたチャンネルの数の分，繰り返す：

i. SPI 送信時に必要な 1byte 目に代入すべきデータを計算し，
変数[first_byte]に代入

```
+---+-------+------------+--------------+------+
| 0b | 00000 | [StartBit] | [SGL/DIFF] | [D2] |
+---+-------+------------+--------------+------+
```

0b : 2 進数開始
00000 : 8bit にするために 0 で補完
1 : Start Bit
1 : SGL/DIFF・・・Input Configuration/[0 or 1]
0 : この部分に D2 の値が入る.
ch>>2：D2 の値

ii. SPI 通信時に必要な 2byte 目に代入すべきデータを計算し，
変数[second_byte]に代入

```
+------+------+--------+
| [D1] | [D0] | 000000 |
+------+------+--------+
```

ch&0b011：D1 と D0 の値
<<6：6 つ「0」を補完

iii. SPI で AD コンバーター[MCP3208]からデータを取得する.
first_byte：1 バイト目の 8bit 0 補完＋開始 bit+制御 bit
second_byte：2 バイト目の 8bit　制御 bit+0 補完
0：3 バイト目の 8bit　0 補完

iv. 現在時刻を取得（記録用）
v. iiiで取得したデータを見やすく加工する.（余分な部分を削除）
rx_data[1]：上位 4bit 不定＋下位 4bit データ
(rx_data[1] & 0b00001111)：下位 4bit を抽出
+rx_data[2]：先程の 4bit に 8bit を足す
->計 12bit を抽出
vi. vで求めたデータを電圧[V]に換算
vii. iv,v,viのデータをディクショナリに保存
viii. viiを返す

5.　観測場所

今回の観測は，東京都小金井市貫井南町 4 丁目 23 番地において実施した．（図 12）

（35°41'50.9"N 139°29'34.5"E）

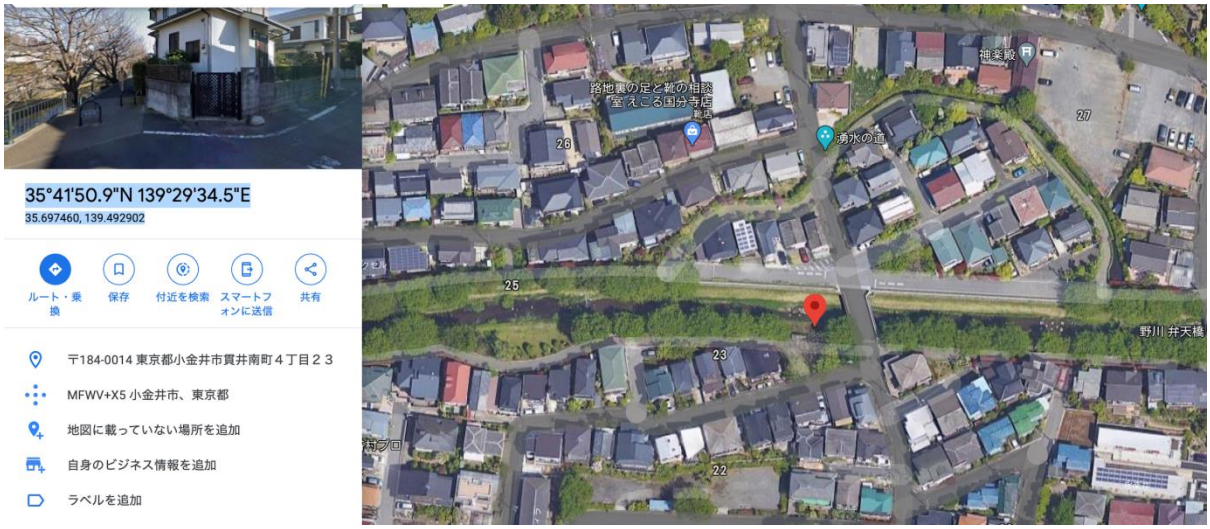


図 12　　観測場所

野川は国分寺市に水源としての湧水を持ち，最終的には多摩川に注ぎ込む一級河川である．この辺りでは，川幅（流水域）が約 4m．水深約 80cm である．

6. 実験結果

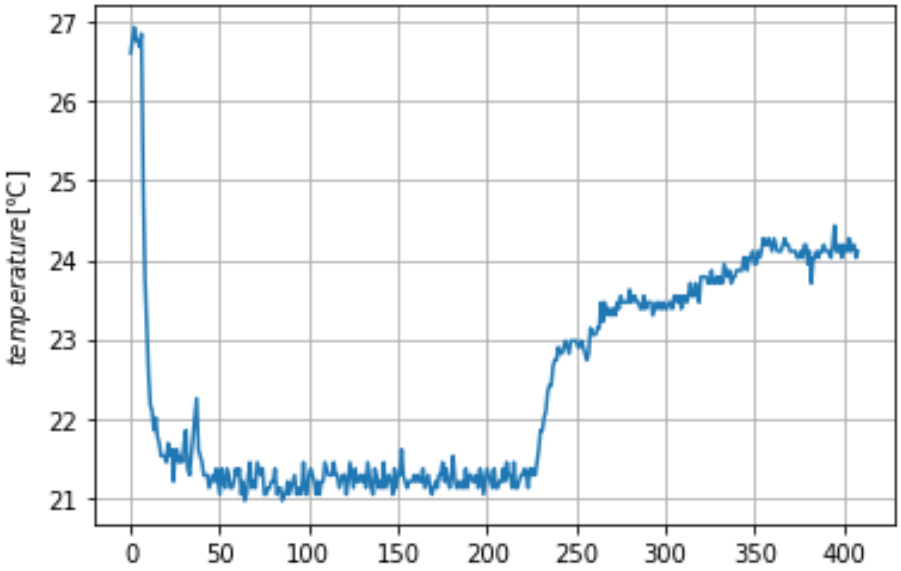


図 13　測定した温度と時間のグラフ

2020/11/2(月)に行った観測の際に，センサーによって得られたデータを温度表示に変換した．そして，その温度表示の時間変化を図 13 に示す．また図 13 は，観測機器を沈める前後の 17:02:57~17:04:49 と観測機器を引き揚げる前後の 17:11:16~17:16:16 の 2 回のデータを合体させて，1 つの図としている．

図 13 に示す測定した温度と時間のグラフから，温度センサーは僅かに数値が上下したがグラフで大まかな傾向を知ることができた．夕方の観測であり時間が経つにつれ水温が下がる傾向が見られると考えていたが，温度はおよそ 400 秒間の測定だったため数値の大きな変化は見られなかった．

7.　考察

およそ 20 秒までは水中に沈めた際の温度低下，225 秒以降の温度上昇は水中から引き上げた際の温度上昇だと考えられる．

温度の数値が細かく±1℃程度上下した理由としては AD コンバーターの分解能が高く（最小分解能 0.122℃）温度センサーの測定誤差の最大値は±1℃であるため，温度センサーからの僅かな電圧の変化も数値に反映されたと考えられる．よって正確な温度を得るためには前後 5 秒～10 秒の数値を平均したほうが良いということが分かった．

装置全体で見ると，この装置の自動で観測・記録をできるという点は，観測の際にも負担軽減として役立ったため環境が厳しい水中を観測する上でとても有用なシステムであると考えられる。

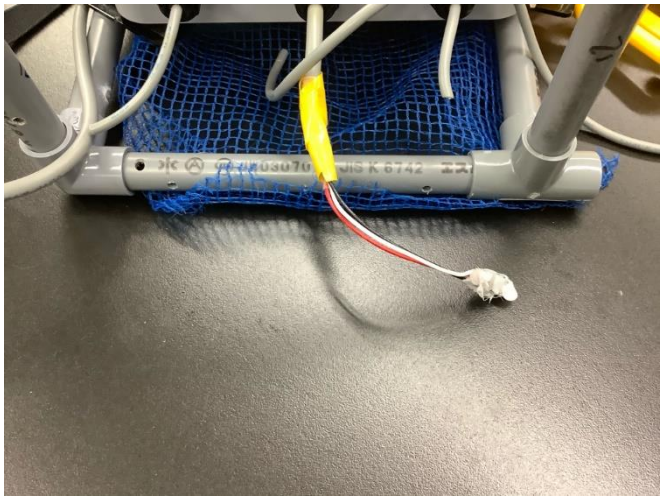


図 14 観測に使用した温度センサー

8. 今後の課題

初期段階では，耐水殻にカメラを搭載し撮影も行う予定だった．しかし，ソフトウェア面の事情により実装を見送りせざるをえなくなった．カメラの撮影を行う Python ライブラリである「OpenCV」がインストールできなかったのだ．原因は以下のように考えられる．RaspberryPi の環境構築に不備があり，Python 関連のディレクトリ構造を把握できていなかったため，apt-get コマンドおよび pip, pip3 を利用した OpenCV のインストール時に問題が発生してしまったことである．

また，[main.py]のメインループ（79 行目～99 行目の，観測・記録・表示を行う部分）では，1 秒間隔（99 行目の time.sleep(1)）でループしていると説明したが，実際は「time.sleep(1)の 1 秒の休み＋観測・記録・表示の処理にかかる時間」でループしているため，1 秒よりも少し長い間隔でループしている．また，観測・記録・表示の処理にかかる時間はその都度処理時間が異なるため，10 ミリ秒程度の差が生じてしまう．（今回の目的であれば）この程度の誤差ならば度外視しても良いと思うが，改善するのであれば，スリープ時間を補修しつつメインとは別のサブスレッドで実行（[threading]などのライブラリを使用）することで改善できる．

まずは，母艦 PC（＝RaspberryPi そのものを操作する PC．RaspberryPi 自体の設定などをした）との連携についてだ．今回のプログラムは，母艦 PC より ssh（または Bluetooth Serial Login）を使いシェルにログインすることによって，遠隔でプログラムを実行したり設定したりした．だが，母艦 PC と RaspberryPi 間の通信が切れてしまうと，RaspberryPi のプログラムが強制終了してしまい（＝SIGHUP），観測を続けることができなくなってしまう．今回は水深が浅かったため，水の影響は少なく母艦<—> RaspberryPi 間の通信は大体安定していたが，実際にもっと深い場所で動かすと，水の影響で電波が減衰してしまい，母艦<—> RaspberryPi 間の通信が切れることは確実である．よって母艦との通信が切れても観測を続けられるようにする必要がある．

改善案としては，このプログラムを継続的にバックグラウンドで起動する，ということが挙げられる．

普通に実行する場合（今回）は，

```
$ python3 main.py
```

だが，nohup コマンド（Linux コマンド）を以下のように使用することで，

```
$ nohup python3 main.py &
```

SIGHUP シグナルを受け取っても無視して，実行し続けることができる．

このように，python プログラムだけでない，周辺環境を整えることが甘かった．

この RaspberryPi を動かしているモバイルバッテリーが，時間が経つと（10 分程度）給電を自動的に止めてしまうという仕様になっている．これは，RaspberryPi のことをモバイルバッテリーが認識していないため，バッテリーがオートオフをしていると考えられる．このオートオフは，バッテリー制御 IC があるため働いていると考えられるので，市販のモバ

イルバッテリーでなく，単品の二次電池（リチウムイオン）を取り付けることでこの問題は解決できると思われる.

この程度の処理ならば RaspberryPi はオーバースペックすぎるようにも感じた．C 言語に移行し高速化・省力化を図るとともに，場合によっては ARM マイコンに移行することも検討したい.

また，RaspberryPi に給電しているバッテリーが突然切れてしまうと，RaspberryPi が強制終了してしまい，RaspberryPi そのものが壊れる原因になってしまう．このようなことが過去に数回あったので，今後は，バッテリーが残量わずかの場合は RaspberryPi をオートシャットダウンする機能も実装したい.

今回用いた AD コンバーター(MCP3208)は 8 チャンネル入力であり，今回製作した基板にはより多くのセンサーを接続できるため，AD コンバーターの性能を最大限に活かせていなかった。それゆえ今後もセンサーの種類と分析を進めたい.

観測機器の筐体に様々なゴミが絡まりにくいように網などを使わないことやバッテリーなどを改良することが出来ると考えた.

そして，水から引き上げた際にセンサーの温度が上昇したのは耐水殻の外に出した温度センサーのホットボンドの隙間から水が侵入した可能性も考えられる．この点においても回路や防水性の再検証が必要だと考えた.

また電子工作の面では，今回の約 6 分の観測では全く問題にはならなかったが，川上から流れてくるさまざまなゴミが機器の網に引っかかることやバッテリーの容量が長期間の観測においては支障をきたす可能性があると考えた.

最終的な目標である「水中の環境を予想する」ことに近づくために，光センサーなどを搭載しカメラの映像も含めて Python の機械学習用ライブラリである Scikit-learn を用いることにより，過去数年分のデータから学習を行い水中の環境を予想できるようにする．そのために，さらに長時間観測することができるようバッテリーを改善し，水中だけでなく外気の環境も観測して比較することによって観測装置の実用化につなげたい。

9. 謝辞

今回の実験および報告書作成などでは田島丈年先生（本校理科非常勤講師・科学部指導員）にはたいへん世話になった．また，報告書は髙松俊介様（早稲田大学基幹理工学部）・大出和輝様（早稲田大学先進理工学部）の両名にも時間が限られた中で丁寧な添削を戴いた．

さらに，本研究は第 9 回気象文化大賞「高校・高専『気象観測機器コンテスト』」として一般財団法人 WNI気象文化創造センターからの助成を受けて行われた．

あわせてここに深謝を申し上げる．

10. 参考文献

"2.7V 4-Channel/8-Channel 12-Bit A/D Converters with SPI Serial Interface," Microchip Technology, https://ww1.microchip.com/downloads/en/DeviceDoc/21298e.pdf, 2020/11/8 .

”LM61 2.7-V, SOT-23 or TO-92 Temperature Sensor,” Texas Instruments Inc., https://www.tij.co.jp/lit/ds/symlink/lm61.pdf?ts=1604837276056&ref_url=https%253A%252F%252Fwww.tij.co.jp%252Fproduct%252Fjp%252FLM61, 2020/11/08

「NJL7502L 照度センサー」新日本無線株式会社, https://www.njr.co.jp/products/semicon/PDF/NJL7502L_J.pdf, 2020/11/8

福田和宏.『Raspberry Pi 電子工作　実践講座　改訂第 2 版』. ソーテックス社.2019,288p

柴田淳.『みんなの Python　第 4 版』. SB クリエイティブ社.2018,484p

---

[i] "2.7V 4-Channel/8-Channel 12-Bit A/D Converters with SPI Serial Interface," Microchip Technology 2008, https://ww1.microchip.com/downloads/en/DeviceDoc/21298e.pdf, p.19.TABLE5-2

[ii] "2.7V 4-Channel/8-Channel 12-Bit A/D Converters with SPI Serial Interface," Microchip Technology 2008, https://ww1.microchip.com/downloads/en/DeviceDoc/21298e.pdf, p.20.FIGURE5-1

[iii] "2.7V 4-Channel/8-Channel 12-Bit A/D Converters with SPI Serial Interface," Microchip Technology 2008, https://ww1.microchip.com/downloads/en/DeviceDoc/21298e.pdf, p.21.FIGURE6-1

[iv] ”LM61 2.7-V, SOT-23 or TO-92 Temperature Sensor,” Texas Instruments Inc.2016, https://www.tij.co.jp/lit/ds/symlink/lm61.pdf?ts=1604837276056&ref_url=https%253A%252F%252Fwww.tij.co.jp%252Fproduct%252Fjp%252FLM61, p.7.

[v] ・「NJL7502L 照度センサー」新日本無線株式会社 2013, https://www.njr.co.jp/products/semicon/PDF/NJL7502L_J.pdf, p.4.

11. 付録

ソースコード

[main.py]

main.py

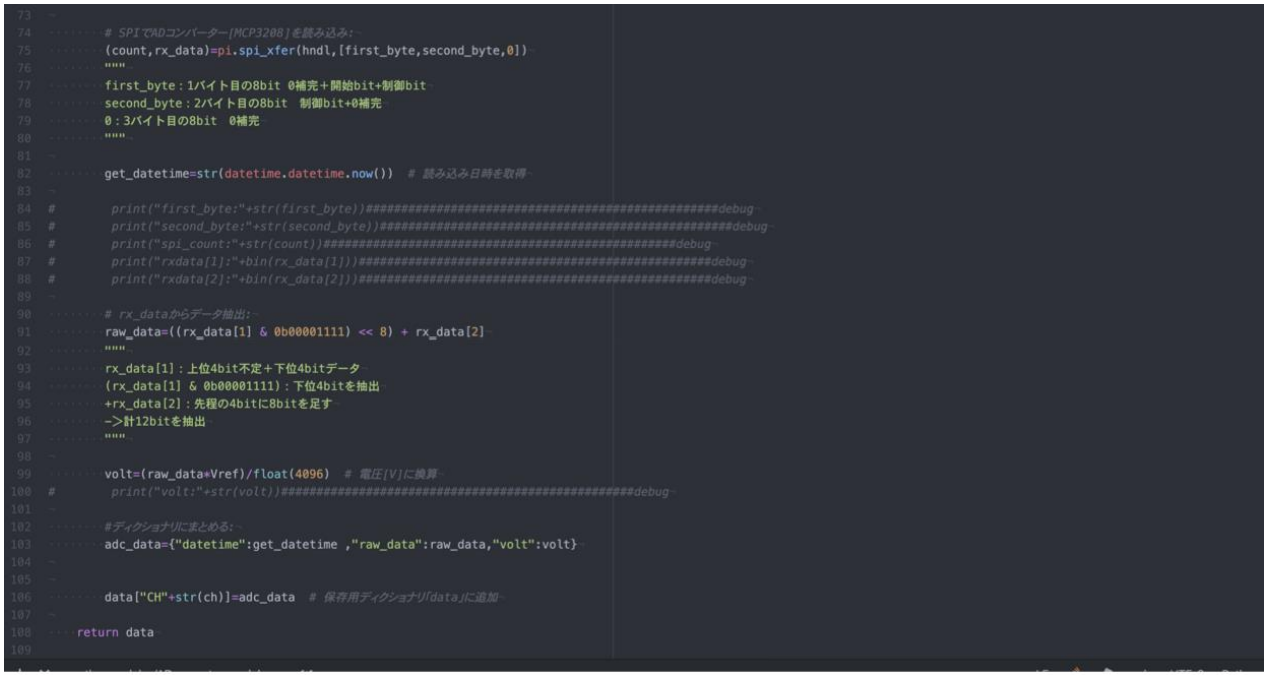
```
"""
このプログラムは、PEP8にできるだけ準拠してください。
PEP8: https://pep8-ja.readthedocs.io/ja/latest/
解説: https://qiita.com/simonritchie/items/bb06a7521ae6560738a7

main.py
"""
# ライブラリインポート:
import pigpio  # GPIO制御ライブラリ
import time  # 時間関係のライブラリ(UNIX時間(エポック秒)を取得)[default]
import datetime # 時間関係のライブラリ(現在日時のdatetimeオブジェクトを取得)[default]
import os  # システム関係を扱うライブラリ[default]
import pathlib  # ファイル/ディレクトリ関係を扱うライブラリ[default]
import csv  # .csvを扱うライブラリ[default]


# 自作モジュールをインポート:
from pythonmodules import sensor_module as sensor
from pythonmodules import ADconverter_module as adc


# 必要なGPIOピンのGPIO番号を変数に入れておく:
#led=2


# 初期設定:
try:
    pi=pigpio.pi()  # pigpioを「pi」で使えるようにする

    sensor.lx_resistance=7.5  # 照度センサに接続する抵抗値を代入(モジュール内変数へのアクセス)

    # SPI初期設定:
    # ADコンバーター[MCP3208]にSPI接続:
    adc.Vref=3.3  # リファンレス電圧(どの程度まで計測可能かを指定)(モジュール内変数へのアクセス)
    spi_channel=0  # SPI接続機器の番号 CEx /0-1 (0-2 for the auxiliary SPI)
    baud=50000  # 通信速度 /32K-125M (values above 30M are unlikely to work)
    spi_flags=0  #spi接続時のオプション /2進数22ビットを10進数に変換して代入

    # open SPI device on channel 0 in mode 0 at 50000 bits per second:
    adc.hndl=pi.spi_open(spi_channel,baud,spi_flags)  # デバイスオープン

    # コネクションが確立してなかったら処理を終える:
    if not pi.connected:
        exit(1)


except AttributeError:  # 例外処理
    print("●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●")
    print("[sudo pigpiod]とコマンドラインで打ってから、このプログラムを実行して下さい")
    print("●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●")
    sys.exit(1)  # プログラム終了(エラー値を返す)

print("start")


# 記録用フォルダ,ファイル作成:
#「OBSERVATION_DATA」という名のディレクトリを、カレントディレクトリに作成(もし既に存在している場合は、スルー):
# /OBSERVATION_DATA:観測データを保存するディレクトリ
pathlib.Path('OBSERVATION_DATA').mkdir(exist_ok=True)
os.chdir("OBSERVATION_DATA")  # カレントディレクトリを「OBSERVATION_DATA」に移動
creation_datetime=str(datetime.datetime.now())  # 現在日時を取得
# 「OBSERVATION_DATA」下に、この処理の記録フォルダ(実行時刻名)、その下に、写真用フォルダを作成:
pathlib.Path(creation_datetime+'/pictures').mkdir(parents=True)
os.chdir(creation_datetime)  # カレントディレクトリを「creation_datetime」に移動

# .txt形式で保存
DATA_txt=pathlib.Path('DATA.txt')
DATA_txt.touch()  # 観測データを記録テキストファイル「DATA.txt」を作成
DATA_txt_a=DATA_txt.open(mode="a")  # 「DATA.txt」を開く(書き込み専用)
# -> DATA_txt_a.write() で書き込み
os.chdir("../..")  # カレントディレクトリを元に戻す(main.pyと同じ層)


# 処理:
run_duration=300 #実行継続時間 /(seconds)
will_end=time.time()+run_duration # 処理終了時刻を計算


while time.time() < will_end:
    data=adc.get_MCP3208([0,1])
#    for i in data:##########################################################debug
#        if not data[i] is None:#############################################debug
#            print(i+":/raw_data="+bin(data[i]["raw_data"])+" /volt="+str(data[i]["volt"])+"V")#################################################debug


    temp={"datetime":data["CH0"]["datetime"] , "temp":sensor.temp_LM61CIZ(data["CH0"]["volt"])}
    DATA_txt_a.write("\n[datetime="+str(temp["datetime"])+","+"temp[°C]="+str(temp["temp"])+"]  ")

    print("datetime="+str(temp["datetime"])+","+" temp="+str(temp["temp"]))
```

```python

    lx={"datetime":data["CH1"]["datetime"] , "lx":sensor.lx_NJL7502L(data["CH1"]["volt"])}
    DATA_txt_a.write("[datetime="+str(lx["datetime"])+","+"lx[Lux]="+str(lx["lx"])+"] ")

    print("datetime="+str(lx["datetime"])+","+" lx="+str(lx["lx"])+"Lux")


    print("++++++++++++++++++++++++++++++")
    time.sleep(1)


# 終了処理:
pi.spi_close(adc.hndl)  # SPIデバイスを閉じる
DATA_txt_a.close()  # テキストを閉じる
pi.stop()
```

[pythonmodules/ADconverter_module.py]

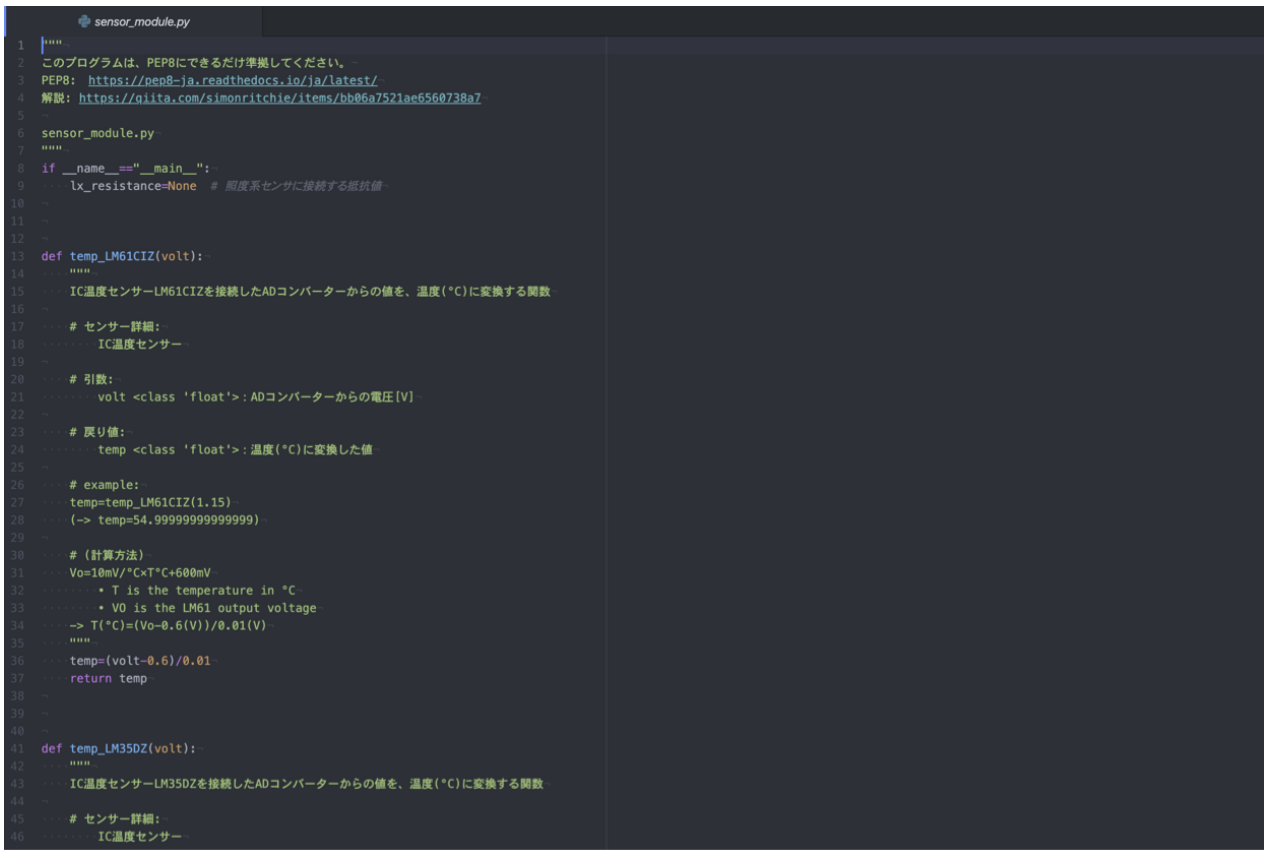
ADconverter_module.py

```python
"""
このプログラムは、PEP8にできるだけ準拠してください。
PEP8:  https://pep8-ja.readthedocs.io/ja/latest/
解説: https://qiita.com/simonritchie/items/bb06a7521ae6560738a7

sensor_module.py
"""
import pigpio  # GPIO制御ライブラリ
import datetime # 時間関係のライブラリ(現在日時のdatetimeオブジェクトを取得)
try:
    pi=pigpio.pi()  # pigpioを「pi」で使えるようにする
except AttributeError:  # 例外処理
    print("●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●")
    print("[sudo pigpiod]とコマンドラインで打ってから、このプログラムを実行して下さい")
    print("●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●")
    sys.exit(1)  # プログラム終了(エラー値を返す)


if __name__=="__main__":
    Vref=None  # リファンレス電圧(どの程度まで計測可能かを指定)
    #main.pyから指定する際は、モジュール内変数へのアクセス表記

    hndl=None  # SPIデバイスオープン用の変数


# 関数群:
def get_MCP3208(adch):
    """
    SPI通信でADコンバーター[MCP3208]から値を読む関数。

    # ADコンバーター詳細:
    2.7V 4-Channel/8-Channel 12-Bit A/D Converters with SPI Serial Interface

    # 引数詳細:
    adch <class 'list'>:A/Dコンバーター[MCP3208]の読み取りたいチャンネル(CHx)一覧
    (CHx,x=0-7)

    # 戻り値詳細:
    data <class 'dict'>:A/Dコンバーターから得られた未加工値一覧（取得しなかったCHは「None」）

    # exaple:
    data=get_MCP3208([0,3])
    (-> data={"CH0":{"datetime":get_datetime ,"raw_data":raw_data,"volt":volt},"CH1":None,"CH2":None,"CH3":{"datetime":get_datetime ,"raw_data":raw_data,"volt":volt},"CH4":N
    """

    # 取得データ保存用のリストを作成
    data={"CH0":None,"CH1":None,"CH2":None,"CH3":None,"CH4":None,"CH5":None,"CH6":None,"CH7":None,}

    for ch in adch:
        first_byte=(0b00000110+(ch>>2))  # 1バイト目:コマンド部分
        """
        +----+-------+------------+------------+-----+
        | 0b | 00000 | [StartBit] | [SGL/DIFF] | [D2]|
        +----+-------+------------+------------+-----+
        0b : 2進数開始
        00000 : 8bitにするために0で補完
        1 : Start Bit
        1 : SGL/DIFF・・・Input Configuration/[0 or 1]
        0 : この部分にD2の値が入る。
        ch>>2:D2の値
        """

        second_byte=(ch&0b011) << 6  # 2バイト目
        """
        +------+------+--------+
        | [D1] | [D0] | 000000 |
        +------+------+--------+
        ch&0b011:D1とD0の値
        <<6:6つ「0」を補完
        """
```



```
        # SPIでADコンバーター[MCP3208]を読み込み:
        (count,rx_data)=pi.spi_xfer(hndl,[first_byte,second_byte,0])
        """
        first_byte：1バイト目の8bit 0補完＋開始bit+制御bit
        second_byte：2バイト目の8bit  制御bit+0補完
        0：3バイト目の8bit  0補完
        """

        get_datetime=str(datetime.datetime.now())  # 読み込み日時を取得

#        print("first_byte:"+str(first_byte))##################################################debug
#        print("second_byte:"+str(second_byte))################################################debug
#        print("spi_count:"+str(count))########################################################debug
#        print("rxdata[1]:"+bin(rx_data[1]))###################################################debug
#        print("rxdata[2]:"+bin(rx_data[2]))###################################################debug

        # rx_dataからデータ抽出:
        raw_data=((rx_data[1] & 0b00001111) << 8) + rx_data[2]
        """
        rx_data[1]：上位4bit不定＋下位4bitデータ
        (rx_data[1] & 0b00001111)：下位4bitを抽出
        +rx_data[2]：先程の4bitに8bitを足す
        ->計12bitを抽出
        """

        volt=(raw_data*Vref)/float(4096)  # 電圧[V]に換算
#        print("volt:"+str(volt))##############################################debug

        #ディクショナリにまとめる:
        adc_data={"datetime":get_datetime ,"raw_data":raw_data,"volt":volt}


        data["CH"+str(ch)]=adc_data  # 保存用ディクショナリ「data」に追加

    return data
```

[pythonmodules/sensor_module.py]



sensor_module.py

```
"""
このプログラムは、PEP8にできるだけ準拠してください。
PEP8:  https://pep8-ja.readthedocs.io/ja/latest/
解説: https://qiita.com/simonritchie/items/bb06a7521ae6560738a7

sensor_module.py
"""
if __name__=="__main__":
    lx_resistance=None  # 照度系センサに接続する抵抗値


def temp_LM61CIZ(volt):
    """
    IC温度センサーLM61CIZを接続したADコンバーターからの値を、温度(°C)に変換する関数

    # センサー詳細:
        IC温度センサー

    # 引数:
        volt <class 'float'>：ADコンバーターからの電圧[V]

    # 戻り値:
        temp <class 'float'>：温度(°C)に変換した値

    # example:
    temp=temp_LM61CIZ(1.15)
    (-> temp=54.99999999999999)

    # (計算方法)
    Vo=10mV/°C×T°C+600mV
        • T is the temperature in °C
        • VO is the LM61 output voltage
    -> T(°C)=(Vo-0.6(V))/0.01(V)
    """
    temp=(volt-0.6)/0.01
    return temp


def temp_LM35DZ(volt):
    """
    IC温度センサーLM35DZを接続したADコンバーターからの値を、温度(°C)に変換する関数

    # センサー詳細:
        IC温度センサー
```

```
    # センサー詳細:
        IC温度センサー

    # 引数:
        volt <class 'float'>:ADコンバーターからの電圧[V]

    # 戻り値:
        temp <class 'float'>:温度(°C)に変換した値

    # example:
    temp=temp_LM61CIZ()
    (-> temp=)#########FROM_HERE

    # (計算方法)
    Vout = 10 mv/°C × T
        • Vout is the LM35 output voltage
        • T is the temperature in °C
    -> T(°C)=Vout/(10/1000)
        =Vout*100
    """
    temp=volt*100
    return temp


def lx_NJL7502L(volts):
    """
    フォトトランジスタNJL7502Lを接続したADコンバーターからの値を、照度(Lux)に変換する関数

    # センサー詳細:
        分光感度特性が人間の視感度特性に近いフォトトランジスタ

    # 引数:
        volt <class 'float'>:ADコンバーターからの電圧[V]

    # 戻り値:
        lx <class 'float'>:照度(Lux)に変換した値

    # example:
    lx=lx_NJL7502L(1.15)
    (-> lx=340.4)

    # (計算方法)
    photocurrent[μA]) = Illuminance[Lux] / 2.22
    I[A] = V[E] / R[Ω]
    -> Illuminance[Lux] = ((V[E] / R[Ω]) * (10**6)) *2.22
    I[A] = V[E] / R[Ω]
    -> Illuminance[Lux] = ((V[E] / R[Ω]) * (10**6)) *2.22
    -> Illuminance[Lux] = ((V[E] /(R[kΩ]*(10**3))) * (10**6)) *2.22
    """
    lx= ((volts/(lx_resistance*(10**3))) * (10**6)) *2.22
    return lx
```

[tempareture_program.py]

測定された水温のデータをリスト化し、グラフにするプログラムである。

```
temperature_program.py
#使用モジュールのインポート
import matplotlib.pyplot as plt
import re

#ファイルから抽出する関数の記述
def extract_text_in_file(filepath, pattern_prev, pattern_next):
    extracted_text_array = []
    pattern = pattern_prev + '(.*)' + pattern_next
    with open(filepath) as f:
        lines = f.readlines()
        for line in lines:
            tmp_extracted_text_array = re.findall(pattern, line)
            extracted_text_array.extend(tmp_extracted_text_array)

    return extracted_text_array

#データが記述されているファイルの指定(要書換え?)
filepath = 'DATA1.txt'

#どこで切るか
pattern_prev = 'temp\[°C\]='
pattern_next = '\]  \['
extracted_text_array = extract_text_in_file(filepath, pattern_prev, pattern_next)

for extracted_text in extracted_text_array:
    print(extracted_text)

#数値のみを記述するファイルを作成・編集
path_w = 'temperature_data_'+filepath

with open(path_w, mode='w') as f:
    for extracted_text in extracted_text_array:
        f.write(extracted_text)
        f.write(" ")

with open(path_w) as f:
    print(f.read())

#データの記述
~/Desktop/temperature_program.py  1:1   LF  UTF-8  Python  GitHub  Git (0)
```

temperature_program.py

```
#データの記述
f = open(path_w)
datas = f.read().split()
print(datas)
f.close()

#リスト化
datas_int_list = [float(s) for s in datas]
print(datas_int_list)

#グラフ記述
x = list(range(len(datas_int_list)))
y = datas_int_list
plt.plot(x,y);
plt.title('data1')
plt.xlabel('$time[s]$')
plt.ylabel('$temperature[℃]$')
plt.grid(True)
plt.show()



#データが記述されているファイルの指定(要書換え?)
filepath2 = 'DATA2.txt'

#どこで切るか
pattern_prev = 'temp\[°C\]='
pattern_next = '\]  \['
extracted_text_array = extract_text_in_file(filepath2, pattern_prev, pattern_next)

for extracted_text in extracted_text_array:
    print(extracted_text)

#数値のみを記述するファイルを作成・編集
path_w = 'temperature_data_'+filepath2

```

~/Desktop/temperature_program.py 60:1 LF UTF-8 Python GitHub Git (0)

temperature_program.py

```
with open(path_w, mode='w') as f:
    for extracted_text in extracted_text_array:
        f.write(extracted_text)
        f.write(" ")

with open(path_w) as f:
    print(f.read())

#データの記述
f = open(path_w)
datas2 = f.read().split()
print(datas2)
f.close()

#リスト化
datas2_int_list = [float(s) for s in datas2]
print(datas2_int_list)

#グラフ記述
x = list(range(len(datas2_int_list)))
y = datas2_int_list
plt.plot(x,y);
plt.title("data2")
plt.xlabel('$time[s]$')
plt.ylabel('$temperature[℃]$')
plt.grid(True)
plt.show()



#データ結合
datas=datas_int_list+datas2_int_list



#グラフ記述
x = list(range(len(datas_int_list)+len(datas2_int_list)))
y = datas
plt.plot(x,y);
```

~/Desktop/temperature_program.py 60:1 LF UTF-8 Python GitHub Git (0)

```
#グラフ記述
x = list(range(len(datas_int_list)+len(datas2_int_list)))
y = datas
plt.plot(x,y);
plt.title("both_datas")
plt.xlabel('$time[s]$')
plt.ylabel('$temperature[℃]$')
plt.grid(True)
plt.show()

```

~/Desktop/temperature_program.py 60:1 LF UTF-8 Python GitHub Git (0)