ICARUS Mark.Ⅱを使用した
高高度撮影および大気汚染度の測定
と気象データの回収

香川高等専門学校　宇宙開発研究部 A 班

二次審査報告書

開発者

電気情報工学科　3 年　　平山覚也

電気情報工学科　2 年　　土井大地

機械工学科　　2 年　　浪越柊弥

担当教員

電気情報工学科　准教授　村上幸一

目次

第 1 章　緒論

1-1 実験背景

香川高専高松キャンパスでは，千葉工業大学と九州大学の協力で，様々な装置を搭載したバルーンサット（上空 30km 付近まで上昇する気球）を放球し，上空の大気からデータを得る実験や，通信実験などを行っている。

昨年度，宇宙開発研究部では環境に対する深刻な問題である大気汚染を，上空の汚染度という観点から調査するため，大気回収装置「ICARUS」を搭載したバルーンサットを放球し，回収した大気のサンプルを解析する実証実験を行なった。

この実証実験を通し，大気の一部を含んでいると思われるサンプルの解析で得た微粒子の大きさと数から，含まれている汚染物質の種類をある程度予想することに成功した。しかし，装置のプログラムがうまく働かなかったほか，大気回収機構の欠陥が露呈する等数多くの課題を残す結果となった。また，本来，この装置には上空の気象データを回収するセンサの搭載なども視野に入れていたが，実験日に間に合わないなどの理由で断念している。

そこで，今回は前回の「ICARUS」から構造を大きく見直した大気回収装置「ICARUS Mark.Ⅱ」を開発し，より純粋な上空の大気サンプルの回収と，センサによる上空の気象データ取得を目的として，昨年に引き続き 2 回目の実証実験を行うこととした。

1-2 既存の大気計測装置

昨年の報告書と同様に，既存の大気計測装置の解説及びデメリットを記載し，今回の計測の有用性について説明する。

1-2-1　フッ素化合物計測器

フッ素化合物は、フッ素化合物を多く含む原料を使用するアルミニウム精錬工場、肥料工場、窯業などの工場排ガスを発生源とする大気汚染物質である。測定方法はイオン電極法と吸光光度法の 2 種類ある。

イオン電極法では、大気中に含まれるガス状無機フッ素化合物を、緩衝液中に捕集溶解させ、この溶液中のフッ素イオン濃度をイオン電極法によって測定し、大気中のガス状無機フッ素化合物濃度を 1 時間、あるいは 3 時間を周期として記録する。

また吸光光度法では、大気中に含まれるガス状無機フッ素化合物を吸収発色液中に捕集溶解させ、この溶液中のフッ素イオン濃度を吸光光度法によって測定する。

1-2-2　塩素化合物計測器

塩化水素ガスは、350ppm 程度の濃度でも人体への害が懸念されており大気汚染防止法で有害物質として区分されている。測定方法は、イオン電極法である。

イオン電極法は、塩化水素ガスの水にきわめて溶けやすく、水分の存在のもとで多くの金属と活発に反応する特性を利用して、焼却炉も含む排ガス中塩化水素ガス測定器は湿式の装置であり、塩素イオン選択電極を検出器とした方式が主なものである。

1-2-3　硫化水素計測器

硫化水素は、製紙やパルプ工業、石油精製、石油化学工業など、広い分野で関連があり、有毒でかつ悪臭の強いガスである。測定方法は、試験紙光電光度法、非分散形赤外線吸収法などがある。

試験紙光電光度法は、酢酸鉛を含浸させたテープ状の試験紙に、試料大気を一定の流速で通過させ、硫化水素と反応して生ずる硫化鉛による褐色の濃淡の程度を光電的に測定し、硫化水素濃度を測定する方法である。

非分散形赤外線吸収法は、硫化水素を二酸化硫黄に変換するコンバータと非分散形赤外線吸収法の二酸化硫黄計測器とを組み合わせ、さらにサンプルスイッチング方式を採用し極微量の硫化水素を測定する方式である。

1-2-4　アンモニア計測器

アンモニアは、悪臭防止法によって、大気中濃度の許容限度が定められている有害な物質である。測定方法は、紫外吸光光度法、化学発光法などが挙げられる。

紫外吸光光度法は、アンモニアのスペクトルのうち、透過率の極小値と極大値から得られる光強度変調信号を利用した測定方法である。この光強度変調信号の振幅が、透過光の吸収の強さに比例することを利用してアンモニア濃度を測定する。

化学発光法は、特殊金属その他の還元触媒を用いて、排ガス中に共存する NOx とアンモニアを還元反応させ、その結果減少した NOx 濃度を化学発光法一酸化窒素計測器で検出し、等価的にアンモニア濃度として測定するものである。

1-2-5　試料非吸引方式による排ガス計測器

SOx や NOx と呼ばれる排ガスは大気汚染の原因となる有害物質である。測定方法は、波長非分散法、波長分散法が挙げられる。 波長非分散法は、 試料ガスに対し測定対象成分の特異な吸収波長に合わせた半導体レーザなどからの単色光を照射し、 その透過光の吸収量を測定し濃度を連続的に求める方法である。 測定成分は、一酸化窒素、 一酸化二窒素、 一酸化炭素、二酸化炭素、アンモニア、塩化水素、メタン、水分がある。 波長分散法は、 試料ガスに対しキセノンランプなどの光を照射し、その透過光を分光して、測定対象成分の特異な吸収波長光を検出する事により濃度を連続的に求める方式である。測定成分は、二酸化硫黄、一酸化窒素、 一酸化二窒素、 二酸化窒素、一酸化炭素、二酸化炭素、アンモニア、塩化水素、メタン、水分がある。

1-2-6　バーティクルカウンタ

空気中や液体中にある埃や不純物などの微粒子を計数する計測器である。 微粒子計測器の流路を通過する微粒子にレーザ光が当たると散乱光を発する。その微粒子からの光の散乱の強さを測り、大きい粒子ほど散乱光は強くなることを利用してその粒子の大きさに比例した光強度を電気信号として読み取り大きさを判定する。そこから粒子数を測定する。

上記の測定は、装置が大きく地上で使うことを目的としているため上空の大気測定を行うことは難しい。そこで上空の大気を地上に持ち帰ることが必要だと考えられる。

1-3 実験目的

1. 上空の大気汚染状況を知るためにバルーンサットを用いて各高度大気を回収する。
2. 持ち帰った大気に含まれた成分を分析するためにバーティクルカウンタを用いて粒径ごとの個数を計測する。
3. バルーンサットの放球から海上での回収までの過程をアクションカメラで撮影し地球の様子を観測する。
4. 湿度・気圧・海面気圧・気温（機体付近に設置したものと湿度センサに搭載されているものの 2 種類）を測定できるセンサにより，上空の気象データの観測及び解析を実施する

第 2 章　実験方法

2-1 大気回収装置の製作

図 1，図 2 に装置の大まかな概要図を示す。

発泡スチロール外部

(1)高さ 13cm 横 24cm 縦 32cm の発泡スチロール箱を用意し、側面すべてに一つずつ、計 4 つ結束バンドが通るような穴をあける。

(2)空気を取り入れるためのチューブが通るような穴を二か所、発泡スチロールの蓋に開けておく。

(3)カメラを使用するため、カメラとカメラを動かし続けるためのバッテリーを発泡スチロールの蓋にガムテープで固定する。

(4)気球につるすために紐を付ける。その際(1)であけた穴から結束バンドを通し、紐を巻き込んで紐を固定させる。

(5)チューブを通すための穴と換気用の穴を 2 つずつ蓋に空ける。このとき，換気用の穴には，防水テープを張り，浸水を防ぐ。

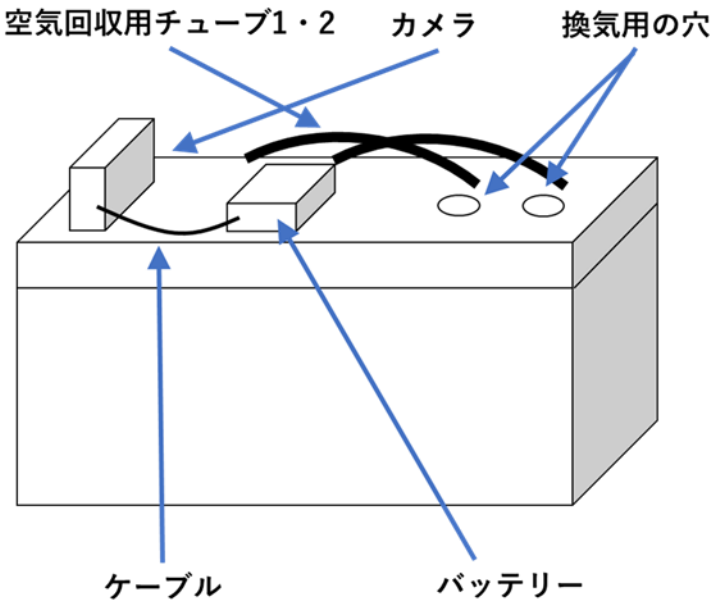


図 1 装置の外観

発泡スチロール内部

(5)RaspberryPi を固定するために小さな発泡スチロール板を用意し結束バンドで

固定する。

(6)RaspberryPi を動かすためのバッテリーを発泡スチロール箱の内側の側面に貼り付けるように設置する。

(7)大気回収用のエアーポンプ二個を瞬間接着剤で固定しそれぞれ二本チューブを取り付ける。

(8)片方は(2)であけた穴に通す。この時長さが足りないため逆止弁を仲介し、チューブを接続する。

(9)もう片方は回収した大気を補完するために逆止弁を仲介し、ジップロックに取り付ける。

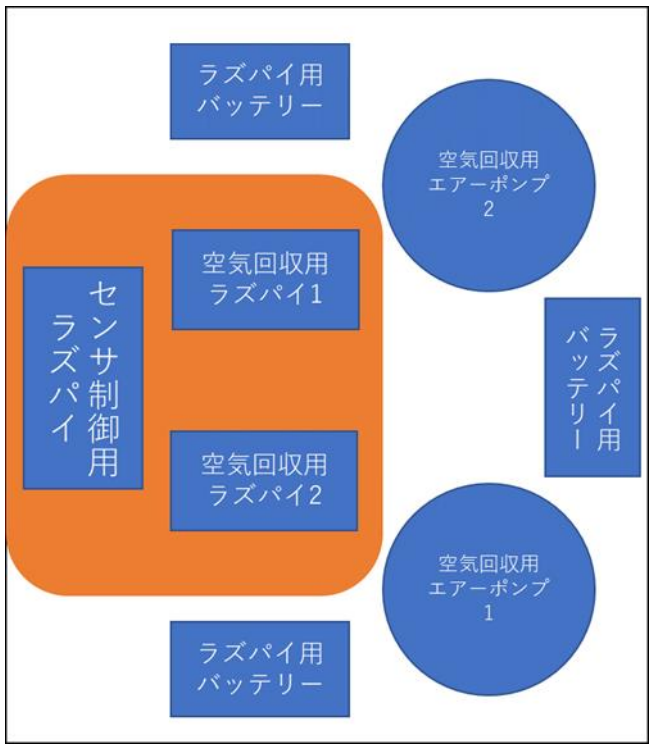


図 2　装置内部

2-2 前回の実験からの改善点・工夫点

(1) 発泡スチロール内部で気球につるすため機体がひっくり返らないように重さの偏りをなくす必要がある。そのため改善点の一つ目としてバッテリーや RaspberryPi の配置を考えた。

(2)前回の実験では、指定した高度に達すると、上空の空気を内部に含んだ注射器の先端を、DC モータを使って蓋を閉じることにより空気を回収するという手法を

とった。しかし、蓋と注射器の間には必ず必ず隙間ができることから完全な密閉ができないまま、注射器を放置して空気を集めてしまった。そのため、いろいろな高度の空気が混ざってしまった。改善点の二つ目としてモータで動くエアーポンプと逆止弁を使った。こうすることによって指定した高度の空気だけを吸引して、逆止弁で空気が逆流しないようになった。

2-3 バルーンサット実験

(1) ペイロード落下時に使用するパラシュートを実験日の風速等を考慮して製作した。
(2) フライトシミュレータで実験日のバルーン経路を確認した。
(3) 発泡スチロールに防水加工を施し、バルーンと発泡スチロールを繋いだ。
(4) バルーン内にヘリウムガスを充填し、放球した。
(5) GPS で追跡しながら回収した。

第 3 章　プログラム

3-1　エアーポンプ制御プログラム

3-1-1　プログラム開発の経緯

ICARUS Mark.Ⅱを制作するにあたり，回収機構を実現させるためのエアーポンプは DC モータで駆動することから，前回 DC モータを駆動させたプログラムを流用する形で記述した。また，前回は全て時間制御でモータを動かしたため，正確な高度の空気を回収できなかったことから，今回大気回収に使用する 2 台のマイコンのうち一方には上空の高度の計測もかねて GPS を搭載し，指定した高度に到達するとモータが動くようプログラムした。

3-1-2　使用した機材の詳細

エアーポンプ制御に使用した機材を表 1 に示す。

表 1　使用機材

使用機器	型式
マイコン	Raspberry Pi3
モータドライバ	TOSHIBA TA8428K JAPAN 834KC1
GPS 受信機キット	GYSFDMAXB
セラミックコンデンサ	0.1［μF］
ジャンパ線	
ブレッドボード	
バッテリー	

次に、モータドライバのブロック図，GPS の回路図および基盤寸法図，端子の説明を図 3，図 4，表 2 に示す。

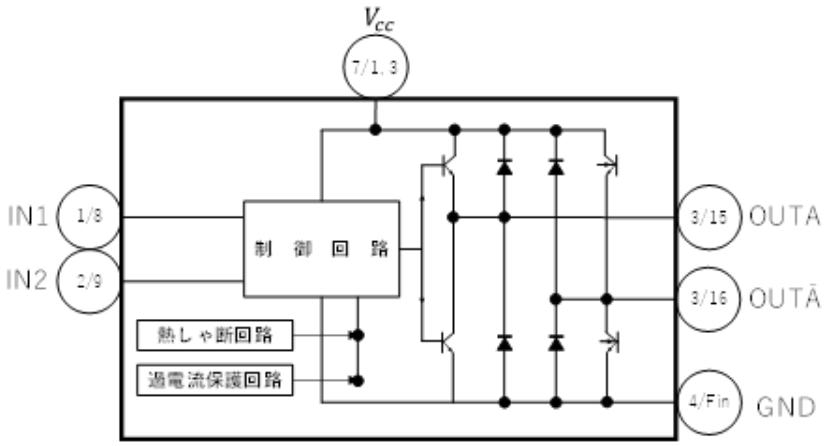
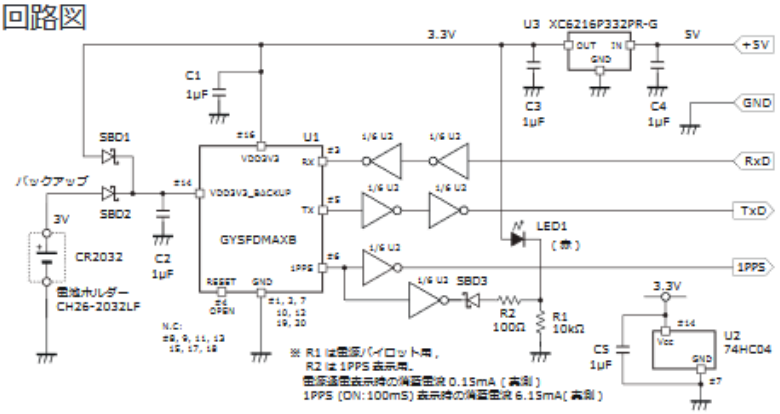


図 3　モータドライバのブロック図

表 2　モータドライバの端子説明

端子番号		端子記号	端子説明
K	FG		
1 2	8 9	IN1 IN2	出力の状態を制御する素子. PNPタイプの電圧コンパレータを内蔵する.
3	15	OUTA	DCモータがつながる端子でSink, SourceともKタイプで1.5A, FGタイプで0.8Aの電流容量をもつ. また, モータの逆起電圧吸収用のダイオードをVcc側とGND側に内蔵している.
4	Fin	GND	接地端子.
5	16	OUT Ā	OUTAピンとの間にモータがつながる端子で, OUTAピンと同様の機能をもつ.
6	他ピン	N.C	Non Connection
7	1, 3	Vcc	電源端子.

◆回路図



◆基板寸法図

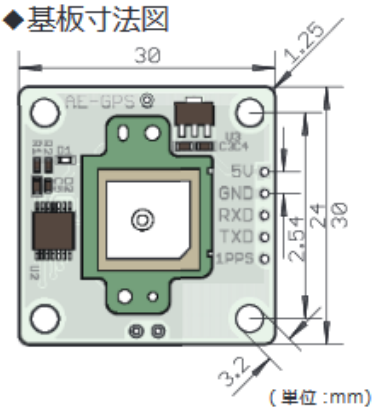


図 4　GPS の回路図および基盤寸法図

3-1-3　事前準備

表 3 にモータドライバの接続図，表 4 に GPS の接続図を示す。

① Raspberry Pi のセットアップ

プログラムを制作する前に Raspberry Pi のセットアップを行い、”sudo apt install python-gpiozero”を入力して gpiozero ライブラリをインストールした。

② Raspberry Pi と機器の接続

(1) Raspberry Pi とエアーポンプの接続をジャンパ線とブレッドボードで表 3 のように行い、モータとジャンパ線をはんだ付けした（GPS を搭載する場合は，モータドライバの IN1 と IN2 を接続する端子を変更した）。また，Raspberry Pi と GPS の接続を表 4 のように行った。

(2) ノイズによる誤動作を防ぐため、モータの両端にセラミックコンデンサをはんだ付けした。

(3) ジャンパ線は直ぐブレッドボードなどから抜けてしまう恐れがあるため、先端を切断して補強パーツに接続するなどの対策を行った。また、Raspberry Pi はモバイルバッテリーから電源を供給した。

③ GPS を使用するための設定

(1) ¥/boot/cmdline.txt を編集し、既存の dwc_ort.lpm_enable をコメントアウトして、コンソールを serial0 から tty1 に変更した。

(2) systemd の設定を変更し、シリアルコンソールを無効化した。

(3) /boot/config.txt を編集し、UART を有効にした後，enable_uart=1 を追加した。

表 3 モータドライバの接続

端子番号	端子記号	説明
1	IN1	GPIO14
2	IN2	GPIO15
3	OUTA	エアーポンプへ
4	GND	GND
5	OUTĀ	エアーポンプへ
6	N。C	未接続
7	Vcc	5V

表 4 GPS の接続

端子番号	端子記号	説明
1	5V	5V
2	GND	GND
3	RXD	GPIO14
4	TXD	GPIO15
5	1pps	GPIO18

3-1-4 プログラムの製作

時間制御のプログラムを図 5，GPS 制御のプログラムを図 6 に示す。

プログラムを図 5、図 6 のように記述し、crontab で起動時実行できるようにした。今回は、20 km 地点の空気を採取する。このとき，GPS を搭載していない時間制御型のプログラムは，約秒速 7 m で上昇する気球を使用することから、現地で気球を準備するまでの 5 分間を考慮し、52 分間後にモータが起動できるように調整した。

```
# -*- coding: utf-8 -*-
from gpiozero import Motor
from time import sleep

### TA8428K moterdriver test ###

motor = Motor(14, 15) # connect GPIO14 to IN1 and GPIO15 to IN2

sleep(3160) # rest for about 52 minutes

motor.forward(1.0) # 1.0 is the speed of the moter

sleep(0.5) # continue to rotate forward for 0.5 seconds


motor.stop()

```

図 5　時間制御のプログラム

```python
#!/usr/local/bin/python
# -*- coding: utf-8 -*-

import serial
import datetime
import os
import csv
from gpiozero import Motor
from time import sleep

motor = Motor(2,3)

ser = serial.Serial(                #みちびき対応ＧＰＳ用の設定
  port = "/dev/ttyS0",            #シリアル通信を用いる
  baudrate = 9600,                  #baudレート
  parity = serial.PARITY_NONE,      #パリティ
  bytesize = serial.EIGHTBITS,      #データのビット数
  stopbits = serial.STOPBITS_ONE,   #ストップビット数
  timeout = None,                   #タイムアウト値
  xonxoff = 0,                      #ソフトウェアフロー制御
  rtscts = 0,                       #RTS/CTSフロー制御
  )
```

```python

#後で使う変数をあらかじめ宣言
alt_lat_long = '0,0,0'     #GPSから得られる、高度、緯度、経度の情
num_sat = '0'   #GPSから得られる、衛星の個数の情報

def sixty_to_ten(x):
  ans = float(int(x) + ((x - int(x)))*100/60)
  return ans

if __name__ == "__main__":

  #GGA用のファイル初期化
  f = open('datagga.csv','w')
  f.write('yyyy-mm-dd HH:MM:SS.ffffff ,a number of satellites ,high ,latitude ,longitude \n')  #出力フォーマット
  f.close()

  #GSV用のファイル初期化
  f = open('datagsv.csv','w')
  f.write('No. ,Elevation in degrees ,degrees in true north \n') #仰角と方位角
  f.close()

  now = datetime.datetime.now()
  if os.path.exists("datagga.csv"):
    new_name = "{0}_{1:%Y%m%d-%H%M%S}.{2}".format("datagga",now,"csv")
    os.rename("datagga.csv",new_name)

  #出力フォーマット
  print ("yyyy-mm-dd HH:MM:SS.ffffff ,a number of satellites ,high ,latitude ,longitude")
  print ("No. ,Elevation in degrees ,degrees in true north \n")
  ################################################################################
```

```
  while True:
    gps_data = ser.readline()  #1行ごとに読み込み、処理を繰り返す
    if not gps_data:
      print("no data")
    #GGA GPSセンサの位置情報を知る
    #$GPGGA,UTC時刻,緯度,緯度の南北,経度,経度の東西,位置特定品質,使用衛星数,
    #水平精度低下率,海抜高さ,高さの単位,ジオイド高さ,高さの単位,DGPS,差動基準地点
    if (gps_data.startswith(b'$GPGGA')): #startswith:1行の先頭文字を検索する
      gpgga = (gps_data.split(b",")) #split:1行をカンマで区切って変数にlist型で保存
      #緯度と経度の情報を、listからfloatに直す
      if gpgga[2]:
        lat_60,long_60,altitude = float(gpgga[2]),float(gpgga[4]),float(gpgga[9])
      else:
        lat_60,long_60,altitude = 0,0,0     #緯度の情報が無い
      #緯度と経度を60進法から10進法に変換、東経と北緯で計算
      if gpgga[3] == "W":  lat_60 *= -1
      if gpgga[5] == "S":  long_60 *= -1
      lat_10,long_10 = sixty_to_ten(lat_60/100),sixty_to_ten(long_60/100)
      #csv形式で出力する用のデータを変数にまとめて保存する(なければ0とする)
      alt_lat_long = "%3.2f,%5.6f,%5.6f" % (altitude,lat_10,long_10) if gpgga[9] else "0,0,0" #高度、緯度、経度
      print(alt_lat_long)
      if (25000.0 < float(gpgga[9]) < 25100.0):
       motor.backward(1.0) # 1.0 is the speed of the moter

       sleep(10.0) # continue to rotate forward for 0.5 seconds

       motor.stop()

```

```
###############
    #GSA 特定タイプを見ることでGPSの通信状況を確認する
    #$GPGSA,UTC時刻,特定タイプ,衛星番号,精度低下率(位置、水平、垂直)
    #if (gps_data.startswith('$GPGSA')):  print gps_data, #特定タイプ(2D,3D等)を確認するために表示。3の時が良好。
###############
    #GSV 受信した衛星の位置等の情報を記録する
    #$GPGSV,UTC時刻,総センテンス数,このセンテンスの番号,総衛星数,
    #衛星番号,衛星仰角,衛星方位角,キャリア/ノイズ比,  を繰り返す
    if (gps_data.startswith(b'$GPGSV')):
      f = open('datagsv.csv','a')
      gpgsv = (gps_data.split(b','))
      #衛星の個数を記録し、情報を追加する
      if (gpgsv[2] == '1'):
        num_sat = gpgsv[3]
        f.write(gpgsv[1] + gpgsv[3] + '\n')
      #それぞれの衛星の番号、仰角、方位角を追加する
      if (len(gpgsv) == 4):  num_sat = '0'
      elif (len(gpgsv) >= 8):
        gsv1 =  gpgsv[4] + gpgsv[5] + gpgsv[6]   #センテンス中一つ目の衛星
        f.write(gsv1.decode('utf-8') + '\n')
      elif (len(gpgsv) >= 12):
        gsv2 =  gpgsv[8] + gpgsv[9] + gpgsv[10]  #二つ目の衛星
        f.write(gsv2 + '\n')
      elif (len(gpgsv) >= 16):
        gsv3 =  gpgsv[12] + gpgsv[13] + gpgsv[14]#三つ目の衛星
        f.write(gsv3 + '\n')
      elif (len(gpgsv) == 20):
        gsv4 =  gpgsv[16] + gpgsv[17] + gpgsv[18]#四つ目の衛星
        f.write(gsv4 + '\n')
      f.close()
```

```
############
    #ZDA NMEA出力における最後の行のため、時間を調べつつ一括ファイル出力する
    #$GPZDA,UTC時刻(hhmmss.mm),日,月,西暦,時,分,
    if (gps_data.startswith(b'$GPZDA')):
      gpzda = (gps_data.split(b","))
      #GPSで取得したUTCの日付を保存する
      yyyymmddhhmmssff = datetime.datetime.strptime(gpzda[4].decode('utf-8') + '/' + gpzda[3].decode('utf-8') +
      '/' + gpzda[2].decode('utf-8') + ' ' + gpzda[1].decode('utf-8'),"%Y/%m/%d %H%M%S.%f")
      time_and_number = "%s,%s" % (yyyymmddhhmmssff,num_sat)
      #ファイル名を書き換える
      #GGAのデータを標準出力、加えてcsvファイルに出力
      f = open(new_name,'a')
      f.write(time_and_number + ',' + alt_lat_long + '\n')
      print(time_and_number + ',' + alt_lat_long)
      f.close()
```

図 6　GPS 制御のプログラム

3-1-5 地上での実行結果

表 5 に，愛媛県愛南町の青い国ホテルでプログラムを起動した際の座標を示す。

時間制御のプログラムは問題なく作動し、既定の時間がたつとエアーポンプを作動させることが出来た。また，GPS を搭載したプログラムは指定した高度になるとエアーポンプが動いたほか，表 5 より，座標が問題なく取得出来ていることが分かる。

表 5　地上で GPS を起動した際に取得した座標

日時	受信した衛星の番号	高さ [m]	緯度	軽度
2022/9/24 21:35	0	28.1	32.95957	132.5608
2022/9/24 21:35	0	28.1	32.95957	132.5608
2022/9/24 21:35	0	28.2	32.95957	132.5608
2022/9/24 21:35	0	28.2	32.95957	132.5608
2022/9/24 21:35	0	28.3	32.95956	132.5608
2022/9/24 21:35	0	28.3	32.95956	132.5608
2022/9/24 21:35	0	28.4	32.95956	132.5607
2022/9/24 21:35	0	28.4	32.95956	132.5607
2022/9/24 21:35	0	28.5	32.95956	132.5608
2022/9/24 21:35	0	28.5	32.95956	132.5608
2022/9/24 21:35	0	28.6	32.95956	132.5608
2022/9/24 21:35	0	28.6	32.95956	132.5608
2022/9/24 21:35	0	28.7	32.95956	132.5608
2022/9/24 21:35	0	28.7	32.95956	132.5607

3-2　センサ制御プログラム

3-2-1　プログラム開発の経緯

ICARUS Mark.Ⅱの制作において，大気回収のほかに上空の気象データを回収するため，湿度・気圧・海面気圧・気温（機体付近に設置した気圧センサのものと湿度センサに搭載されているものの 2 種類。気圧センサのものは今回機体温度の計測に使用する。）を測定できるセンサを搭載した sensorHAT と呼ばれる基盤を使用し，制御するためのプログラムを記述した。また，今回の測定では，気圧と海面気圧から算出できる高度も測定した。これにより，上空の気象データの観測及び解析を実施した。

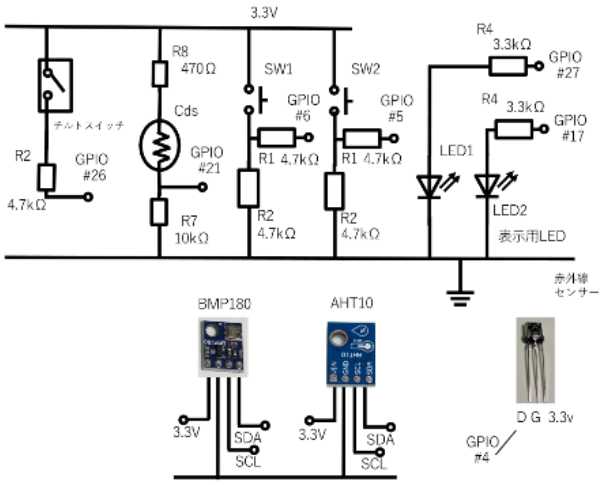
3-2-2　使用した機材の詳細

センサ制御に使用した主な機材を表 6 に示す。

表 6　使用機材

使用機器	型式
マイコン	Raspberry Pi3
気圧センサ	BMP180
湿度センサ	AHT10
バッテリー	
ジャンパ線	

次に，センサの回路図を図 7 に示す。



3-2-3　事前準備

④ Raspberry Pi のセットアップ

(1) git clone https://github.com/momorara/sensorHAT を入力し，リポジトリの sensorHAT をダウンロードした。

(2) i2c を使用できるよう設定しておいた。

(3) センサーが i2c デバイスとして認識できているかどうか，i2cdetect -y 1 コマンドで確認した。このとき，AHT10:38 BMP180:77 が表示されていれば認識できている。

(4) git clone https://github.com/adafruit/Adafruit_Python_BMP.gi を入力し，BMP180 および AHT10 のライブラリをインストールした。

3-2-4　プログラム制作

図 8，図 9 にそれぞれ BMP180 で取得したデータ，AHT10 で取得したデータを csv 形式で保存するプログラムを示す。このプログラムもエアーポンプの制御プログラムと同様に crontab -e でマイコンの起動後に 1 分間隔で実行されるように設定した。

```
27  import Adafruit_BMP.BMP085 as BMP085
28  import datetime
29  import csv
30
31  # Default constructor will pick a default I2C bus.
32  #
33  # For the Raspberry Pi this means you should hook up to the only exposed I2C bus
34  # from the main GPIO header and the library will figure out the bus number based
35  # on the Pi's revision.
36  #
37  # For the Beaglebone Black the library will assume bus 1 by default, which is
38  # exposed with SCL = P9_19 and SDA = P9_20.
39
40  dtime = datetime.datetime.now()
41  date = dtime.strftime('%Y-%m-%d')
42  time = dtime.strftime('%H:%M')
43
44  sensor = BMP085.BMP085()
45
46  temperture = '{0:0.2f} *C'.format(sensor.read_temperature())
47  pressure = '{0:0.2f} Pa'.format(sensor.read_pressure())
48  Altitude = '{0:0.2f} m'.format(sensor.read_altitude())
49  SeaPressure = '{0:0.2f} Pa'.format(sensor.read_sealevel_pressure())
50
51  # Optionally you can override the bus number:
52  #sensor = BMP085.BMP085(busnum=2)
53
54  # You can also optionally change the BMP085 mode to one of BMP085_ULTRALOWPOWER,
55  # BMP085_STANDARD, BMP085_HIGHRES, or BMP085_ULTRAHIGHRES.  See the BMP085
56  # datasheet for more details on the meanings of each mode (accuracy and power
57  # consumption are primarily the differences).  The default mode is STANDARD.
58  #sensor = BMP085.BMP085(mode=BMP085.BMP085_ULTRAHIGHRES)
59  with open('/home/ukk1/ドキュメント/data.csv','a') as f:
60      writer = csv.writer(f)
61      writer.writerow([date,time,temperture,pressure,Altitude,SeaPressure])
62
63  print(temperture)
64  print(pressure)
65  print(Altitude)
66  print(SeaPressure)
67  print(date)
68
```

図 8 BMP180 の制御プログラム

```
import time
# import board
# import busio
# import adafruit_ahtx0
import datetime
import smbus
import Adafruit_BMP.BMP085 as BMP085
import csv

path = '/home/pi/sensorHAT/'

# Create library object using our Bus I2C port
# i2c = busio.I2C(board.SCL, board.SDA)
# print(i2c)

def data_read():
    # Get I2C bus
    #bus = smbus.SMBus(0)  # Rev 1 Pi uses 0
    bus = smbus.SMBus(1) # Rev 2 Pi uses 1
    # when you have a 121 IO Error, uncomment the next pause
    # time.sleep(1) #wait here to avoid 121 IO Error

    config = [0x08, 0x00]
    bus.write_i2c_block_data(0x38, 0xE1, config)
    time.sleep(0.5)
    byt = bus.read_byte(0x38)
    #print(byt&0x68)
    MeasureCmd = [0x33, 0x00]
    bus.write_i2c_block_data(0x38, 0xAC, MeasureCmd)
    time.sleep(0.5)

    data = bus.read_i2c_block_data(0x38,0x00)

    temp = ((data[3] & 0x0F) << 16) | (data[4] << 8) | data[5]
    temp = ((temp*200) / 1048576) - 50
    temp = int(temp*10)/10
    print(temp)
```

```
59
60      humdy = ((data[1] << 16) | (data[2] << 8) | data[3]) >> 4
61      humdy = int(humdy * 100 / 1048576)
62      print(humdy)
63
64      with open('/home/ukk1/ドキュメント/data1.csv','a') as f:
65          writer = csv.writer(f)
66          writer.writerow([temp,humdy])
67
68      return temp,humdy
69
70
71  # i2c接続がない場合のエラーでは何もせずに終了する。
72  try:
73      # sensor = adafruit_ahtx0.AHTx0(i2c)
74
75      # temp = sensor.temperature
76      # humdy = sensor.relative_humidity
77
78      try:
79          temp,humdy = data_read()
80      except:
81          time.sleep(2)
82          temp,humdy = data_read()
83
84      print("Temperature: %0.1f C" % temp, end='', flush=True)
85      print("  Humidity: %0.1f %%" % humdy)
86      # time.sleep(5)
87
88      temp_hosei = 0
89      humdy_hosei = 0
90
91      temp = temp + temp_hosei
92      humdy = str(int(humdy + humdy_hosei))
93
```

```
93
94      dt_now = datetime.datetime.now()
95      s = "摂氏: {0:.1f}"
96      ################## temp ##################
97      # # 最新のデータを一つだけ入れたファイルを作る
98      temp_s = dt_now.strftime("%Y/%m/%d %H:%M.%S") + " :" + s.format(temp) + '\n'
99      with open(path + 'temp_data.txt', mode='a') as f:
100         f.write(temp_s)
101     with open(path + 'temp_data_last.txt', mode='w') as f:
102         s = "{0:.1f}"
103         temp = int(temp * 10)
104         temp_s = str(temp)
105         f.write(temp_s)
106     ####################################
107     ################## humdy ##################
108     # # 最新のデータを一つだけ入れたファイルを作る
109     humdy_s = dt_now.strftime("%Y/%m/%d %H:%M.%S") + "  :" + humdy + '\n'
110     with open(path + 'humdy_data.txt', mode='a') as f:
111         f.write(humdy_s)
112     with open(path + 'humdy_data_last.txt', mode='w') as f:
113         f.write(humdy)
114     ####################################
115
116 # AHTがない場合に温度をBMPから取得する。
117 except:
118     pass
119     print('**** AHT error ****')
120     try:
121         sensor = BMP085.BMP085()
122         temp = sensor.read_temperature()
123         print("Temperature: %0.1f C" % temp, end='', flush=True)
124         print("  Humidity:  no data")
125         dt_now = datetime.datetime.now()
126         temp_s = str(dt_now) + "  :" + str(temp) + '\n'
127         with open(path + 'temp_data.txt', mode='a') as f:
128             f.write(temp_s)
129         with open(path + 'temp_data_last.txt', mode='w') as f:
130             temp = int(temp * 10)
131             temp_s = str(temp)
132             f.write(temp_s)
133
134         # 湿度データを消す
135         with open(path + 'humdy_data_last.txt', mode='w') as f:
136             f.write('0')
137     except:
138         print('**** BMP error ****')
```

図 9　AHT10 の制御プログラム

3-2-5　地上での実験結果

どちらのセンサも問題なく値を取得し，csv 形式で保存することが出来ていた。

第 4 章　装置

今回のバルーンサット実験で使用した大気回収装置内の部品図を以下に示す。

1. エアーポンプ固定具を図 10、11 に示す。この部品は、去年 DC モータの固定に使用したパーツの流用であり，空気を回収するエアーポンプを固定するために使用する。

2. モバイルバッテリーを図 12 に示す。この部品は、エアーポンプと連結させることで実験中の容量確保を行うために使用する。

3. エアーポンプを図 13 に示す。DC モータで駆動しているため，モータドライバを用いることで制御し，空気を吸引・排出できる。

3. ビニル製のチューブを図 14 に示す。このチューブはエアーポンプと連結させることで空気の吸引及び回収袋への運搬を行う。

4. 逆止弁を図 15 に示す。逆止弁は一方向にしか空気を流さず，万が一水が入ってもせき止められることから，チューブに接続することで，空気の逆流や海水による浸水を防ぐために使用する。

5. 空気回収袋を図 16 に示す。空気を完全に密閉して回収するために，密閉性に優れたジップロックを使用した。

6. マイコンを搭載した下駄を図 17 に示す。マイコンを直接機体に固定することは難しいことから，発泡スチロール製レンガを加工して製作し，結束バンドに括り付けることでマイコンを固定した。また，下駄は，足にガムテープを接着することで機体本体と固定した。

7.　実際の機体の内部図を図 18 に示す。

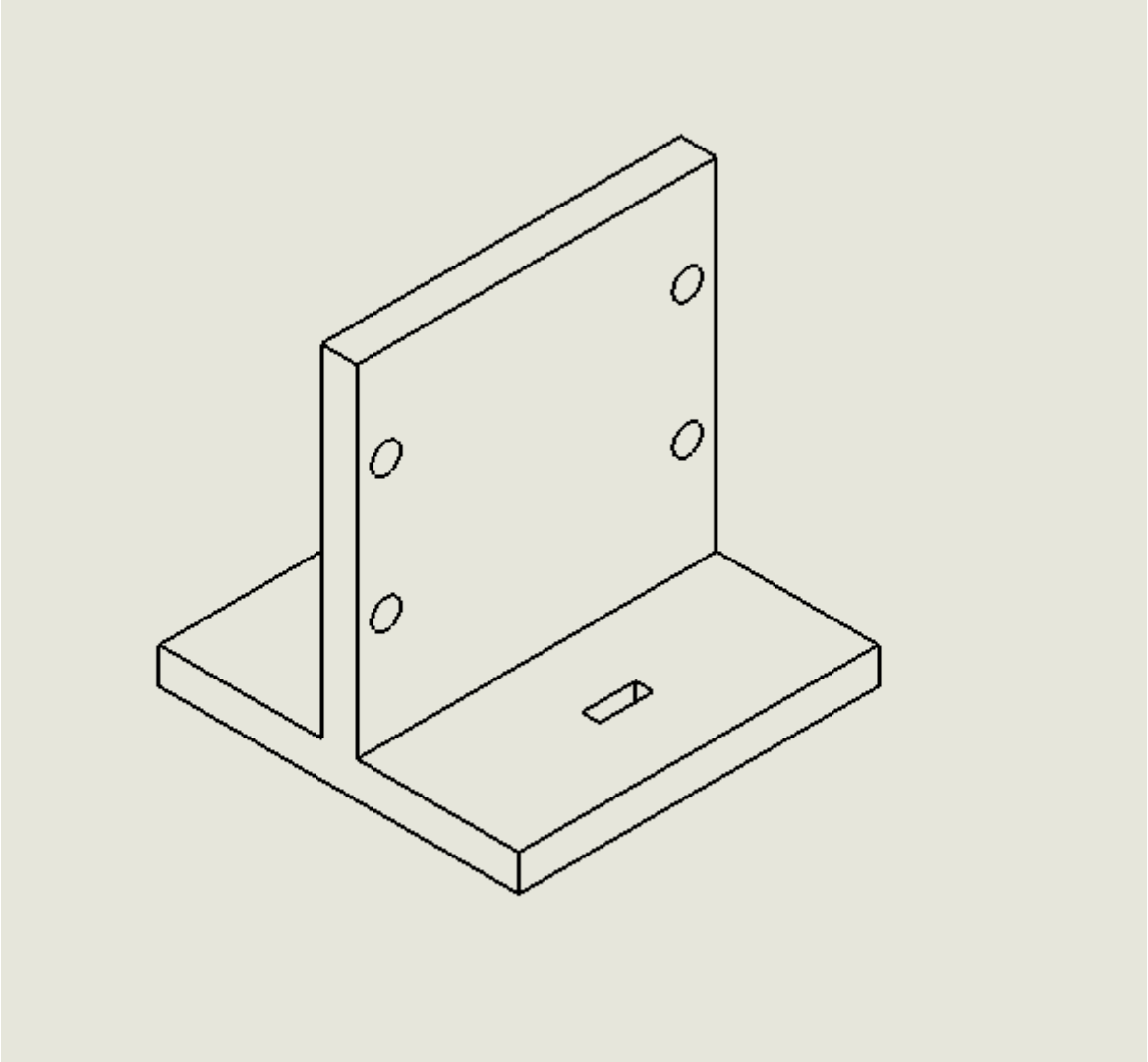


図 10　DC モータ固定具　等角投影図

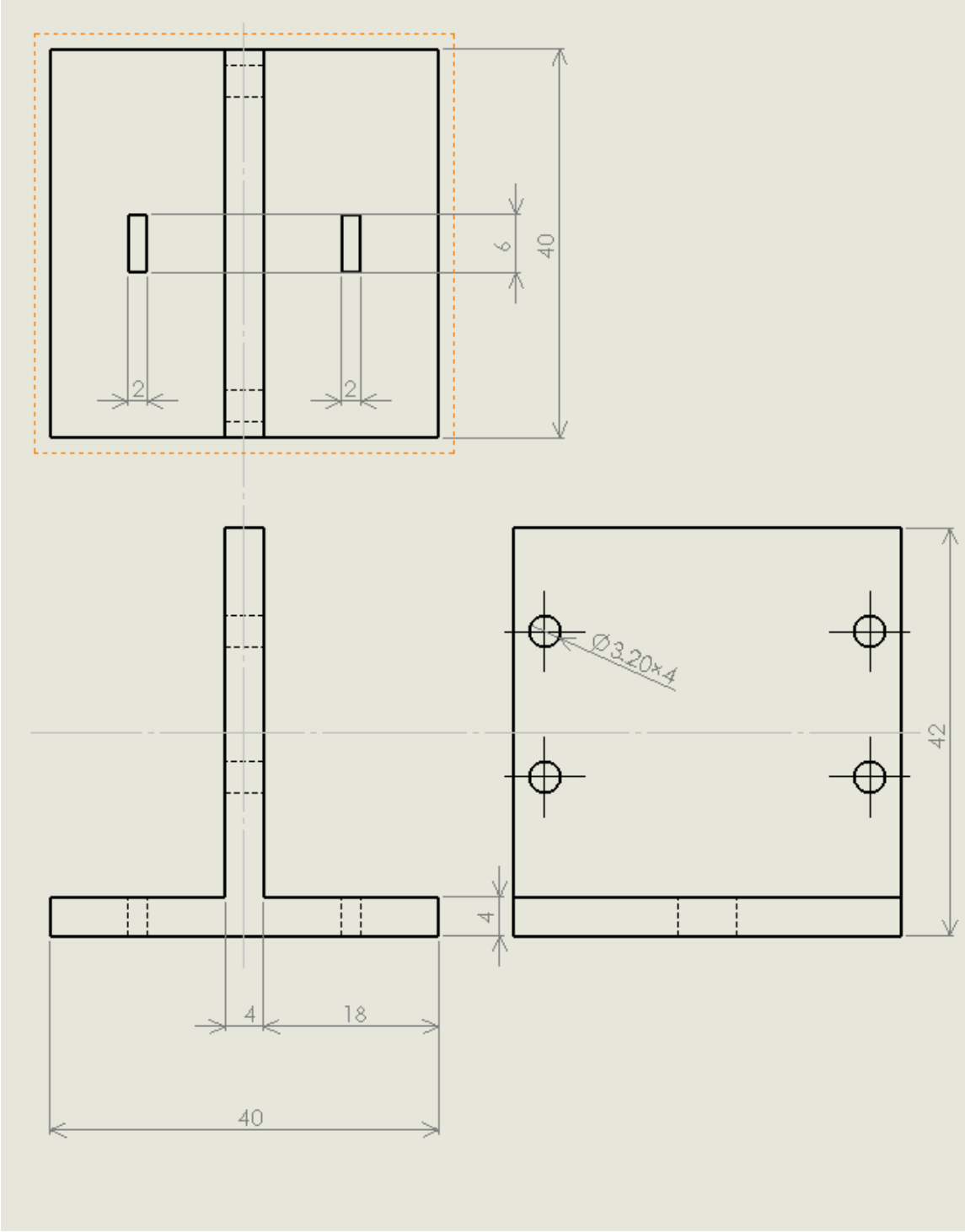


図 11　DC モータ固定具　三面図

図 12　モバイルバッテリー

図 13　エアーポンプ

図 14　ビニル製チューブ

図 15 逆止弁

図 16　空気回収袋及びエアーポンプとの接続図

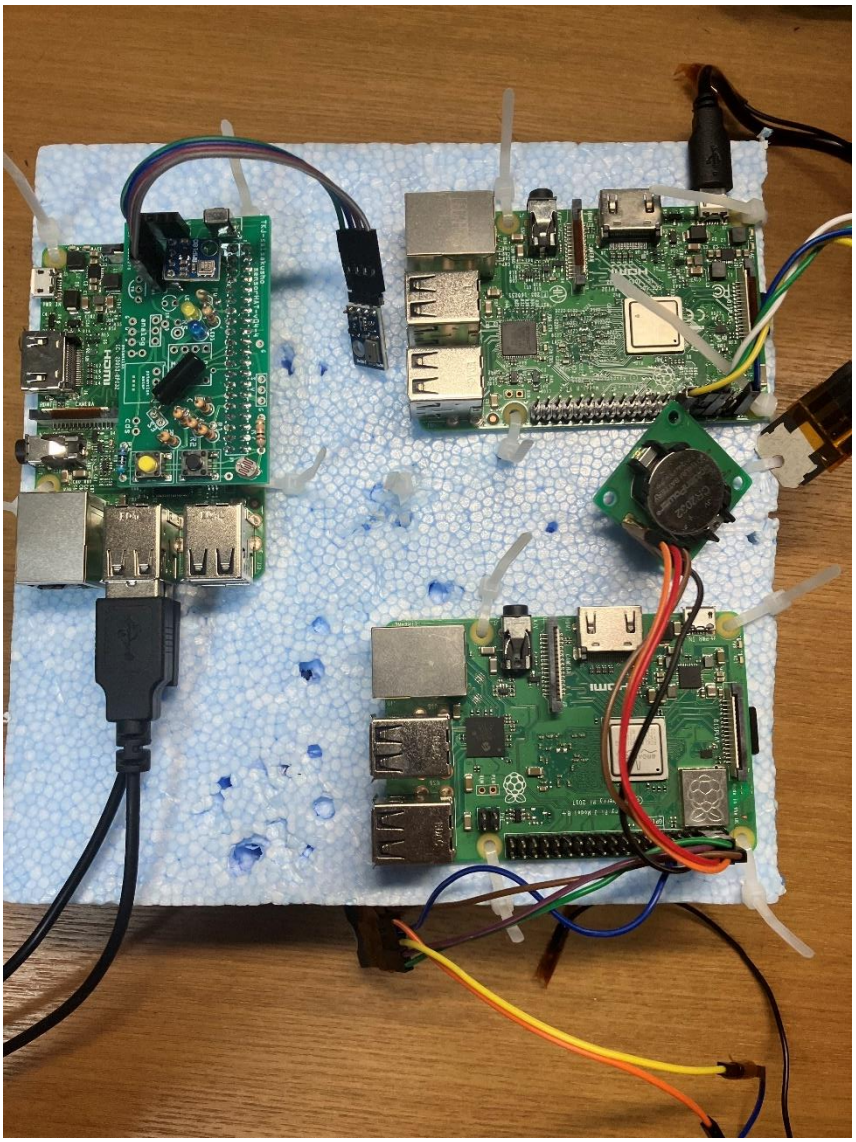


図 17　マイコンを設置した下駄

図 18　機体内部図

第 5 章　実験条件・結果

5-1 実験条件

今回のバルーンサット実験の実験条件を下記に示す。

- 実験日時
  2022 年 9 月 25 日（日）　9:00～
- 実験場所
  放球地点 ： 愛南町の大森山公園駐車場（愛媛県）
  バルーン回収地点 ： 四万十町志和の志和港周辺（高知県）

- シミュレーション経路

  シミュレーションにより得られたバルーンの経路を図 19，20 に示す。

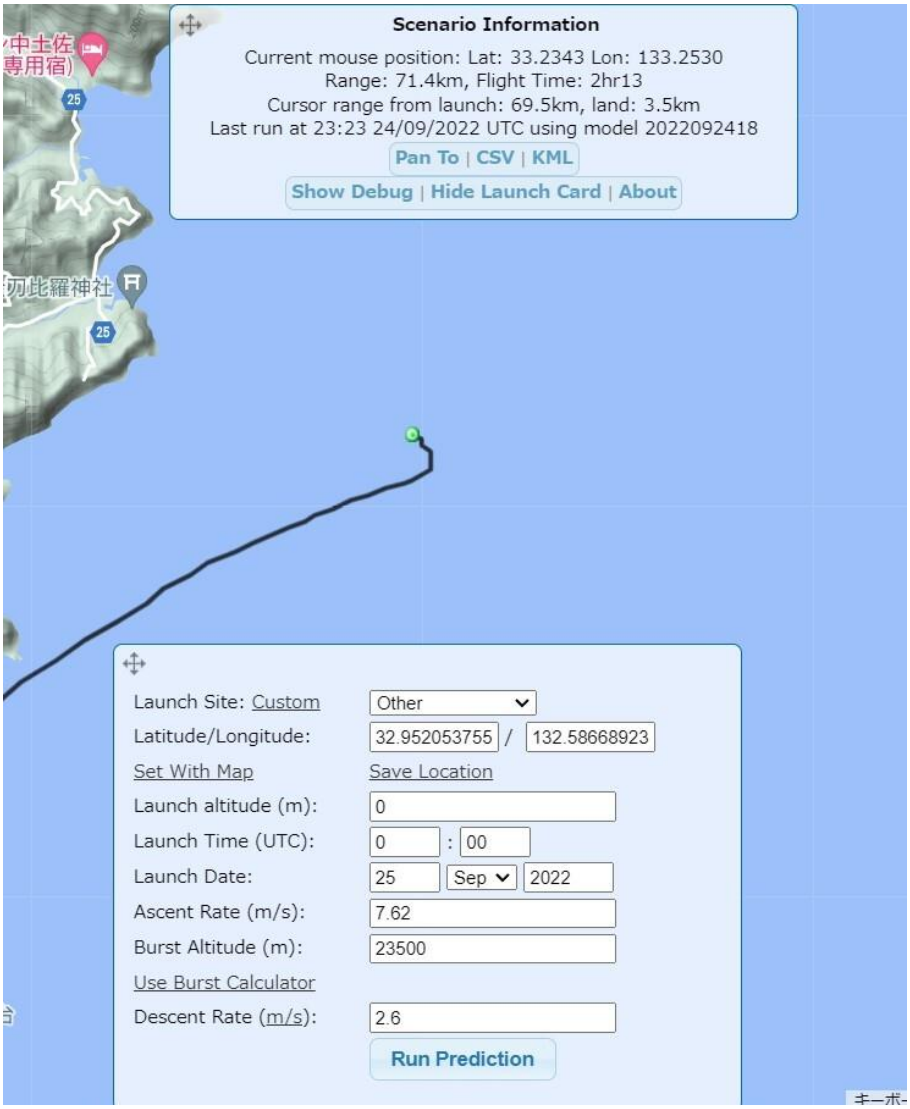


図 19 シミュレーション経路 1

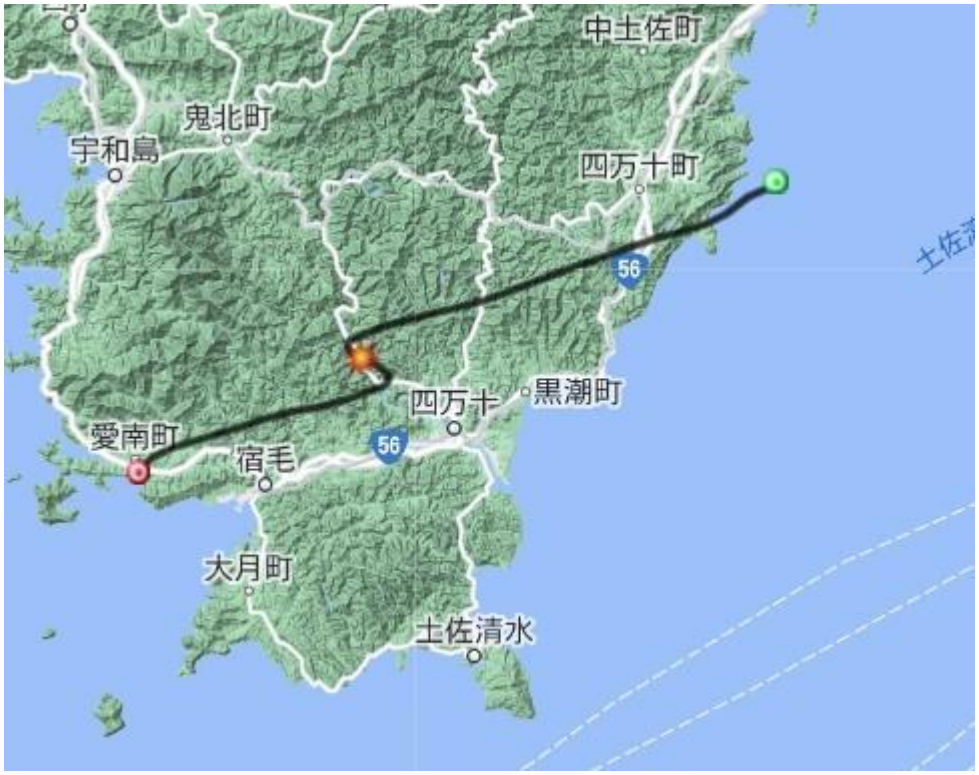


図 20　シミュレーション経路 2

- 気象データ

気象庁のホームページより実験日の気象データを以下に示す。データは和歌山県の潮岬のものである。

また、各高度の風速と風向のデータ(和歌山県潮岬 9 月 25 日 9 時)を表 5-1 に示す。

降水量 ： 0.0mm
平均気温 ： 21.3℃
最低気温 ： 17.5℃
最高気温 ： 25.3℃
日照時間 ： 1.0 時間

表 7　各高度における風速と風向

気圧(hPa)	ジオポテンシャル高度(m)	風速(m/s)	風向き(°)
1000	128	5	58
800	2046	3	263
600	4437	7	251
400	7605	18	265
30	24037	10	100

5-2 実験結果

1. バルーンの放球から回収までの写真を写真 1 から 7 に示す。
   下記に示した写真のようにバルーン実験は放球から回収まで成功させることができた。また，20km 付近の空気を回収することに成功したほか，上空の気象データを取得することにも成功した。しかし，残念ながら GPS により取得した位置情報の記録は，プログラムにバグが発生したことで，途中で中断されるという結果となったほか，カメラのバッテリーが撮影中に落下するというアクシデントがあった。

写真１　放球前の様子

写真２　滞空時の様子(1)

写真３　滞空時の様子(2)

写真 4　滞空時の様子(3)

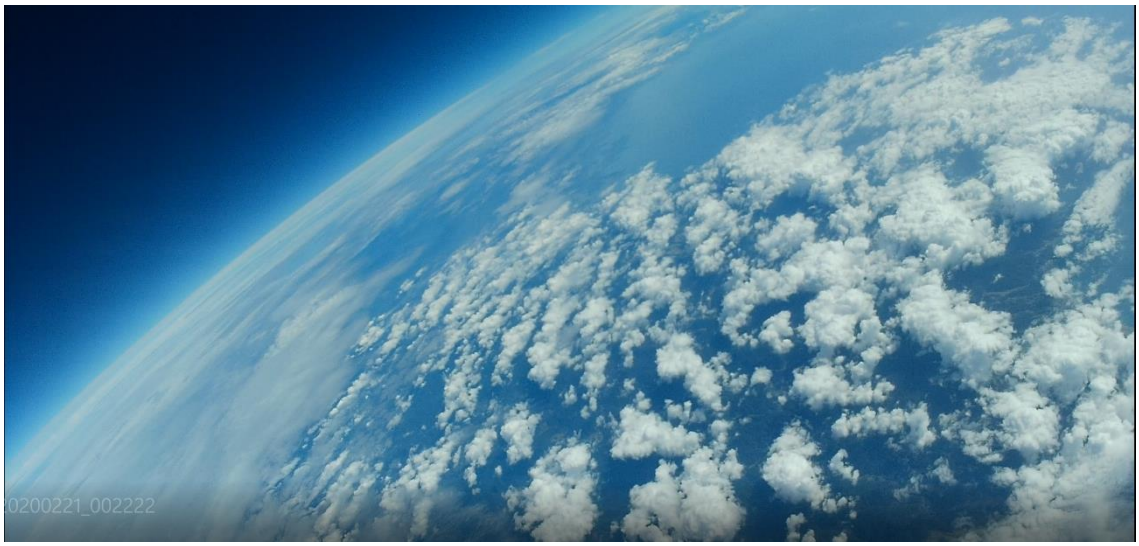


写真 5　滞空時の様子(4)

写真 6　回収した装置の写真

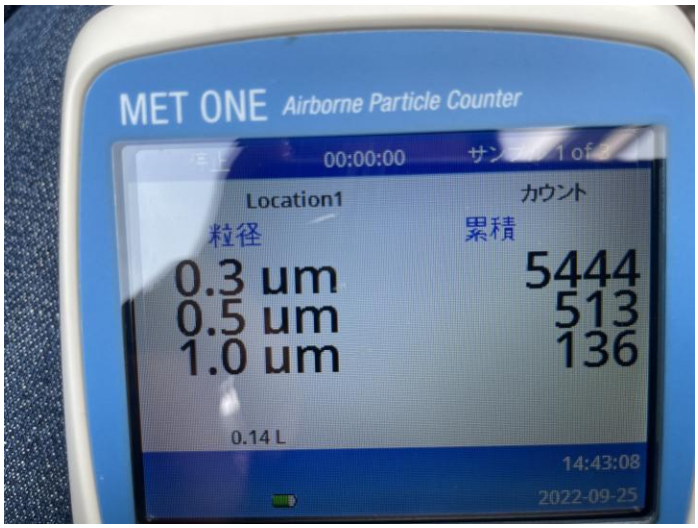


写真 7　大気物質の計測写真

2. 実験の結果バーティクルメータで得られたデータを表 8、図 21、 22、 23 に示す。

表 8　高度ごとの粒子数

高度	粒子数[個]		
	0.3 [μm]	0.5 [μm]	1.0 [μm]
海抜0[m]	7124	767	259
高度20[km]	5667	525	134
上昇50分後（高度20[km]周辺）	5444	513	136

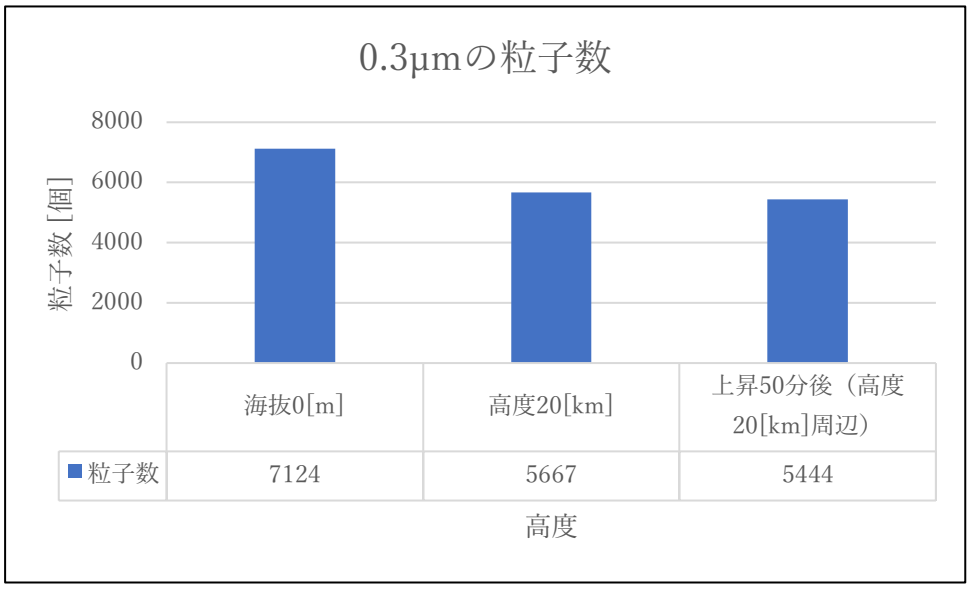


図 21　0.3μm の粒子数

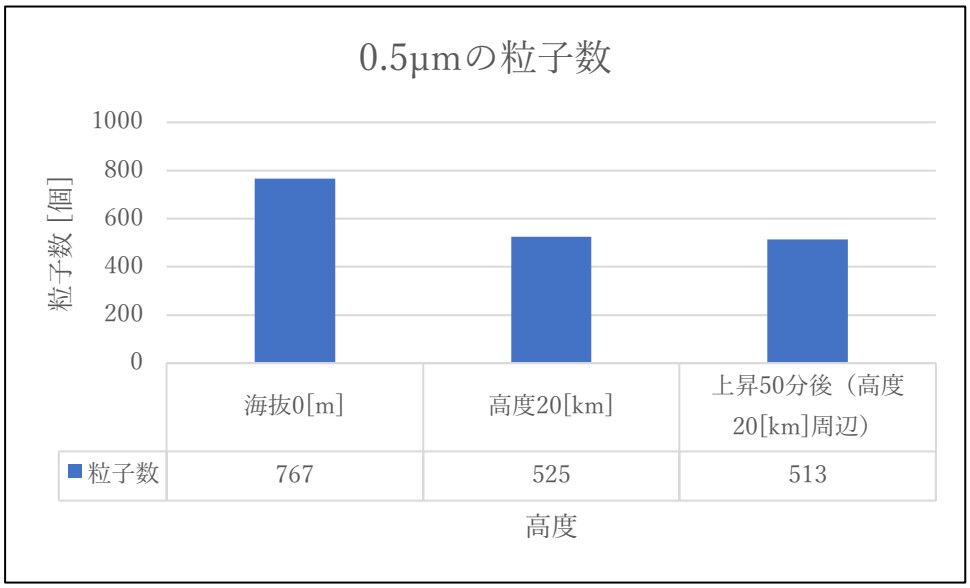


図 22　0.5μm の粒子数

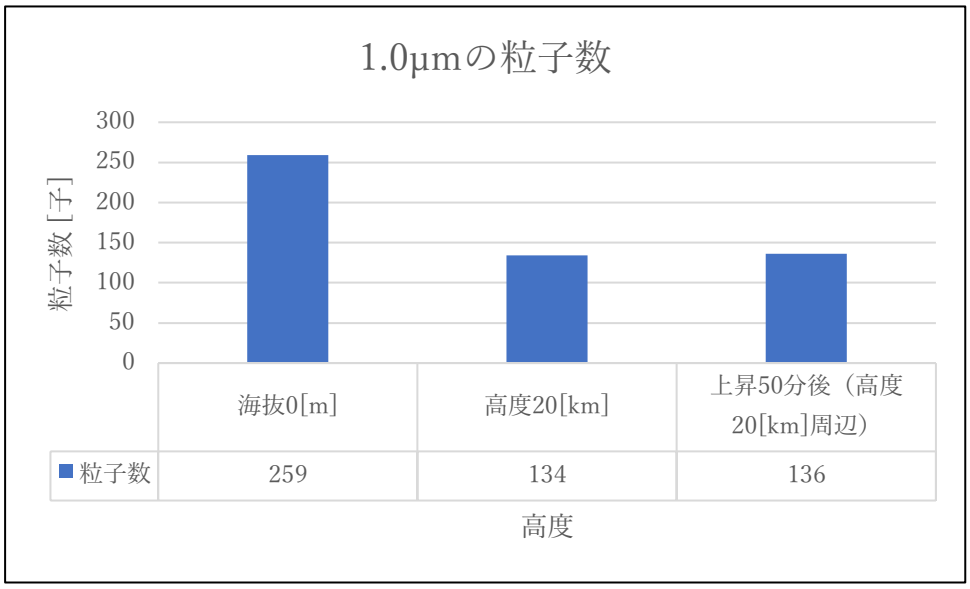


図 23　1.0μm の粒子数

3. センサから得られたデータのグラフを図 24～図 30 に示す。

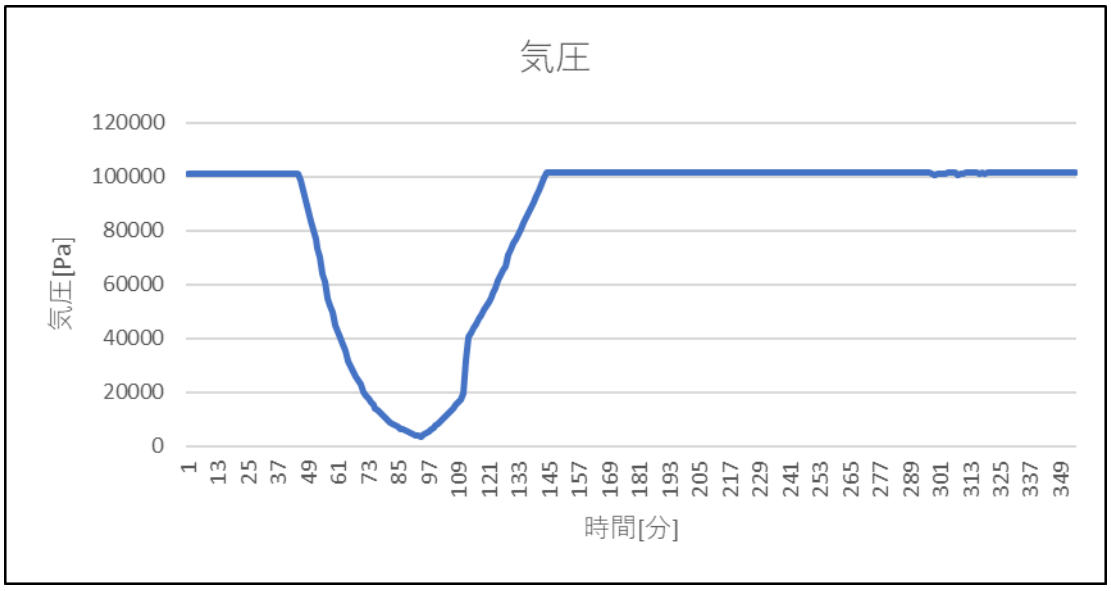


図 24　BMP180 で計測した上空の気圧のデータ

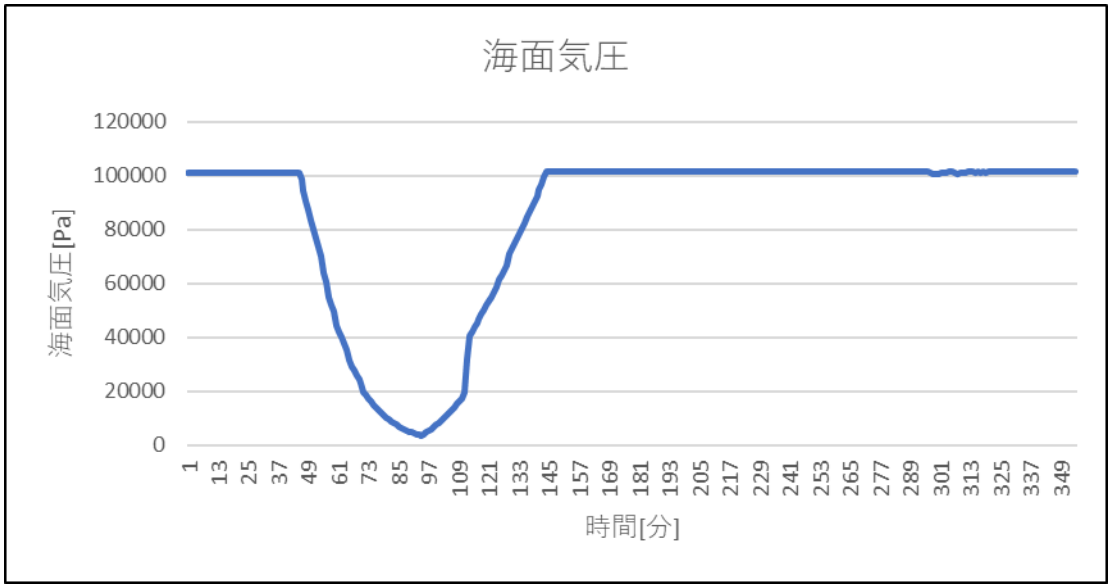


図 25　BMP180 で計測した上空の海面気圧のデータ

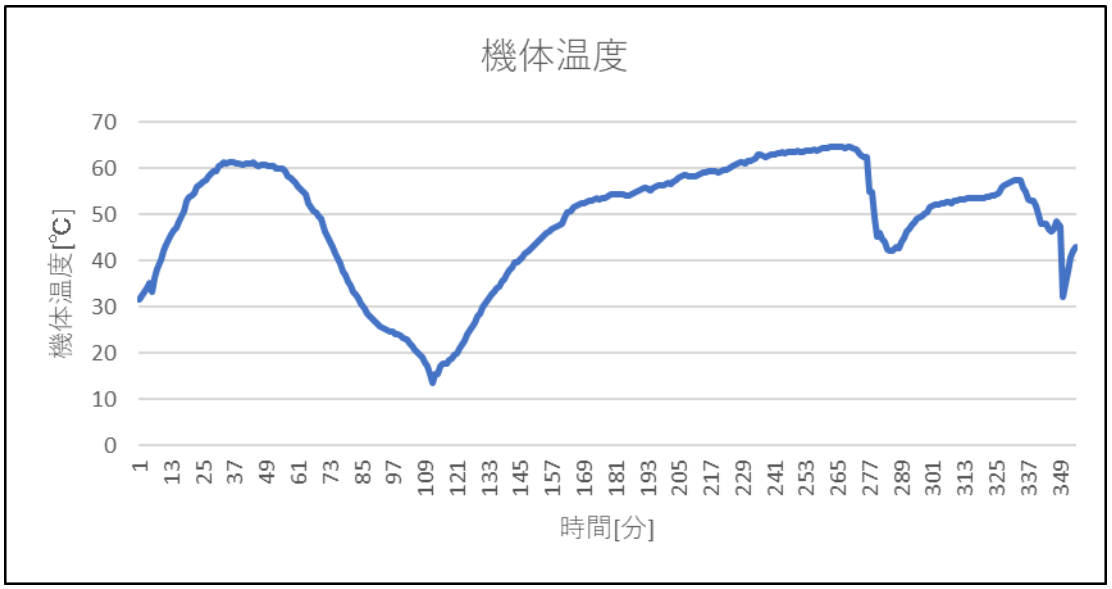


図 26　BMP180 に搭載されている気温センサで計測したマイコンの温度

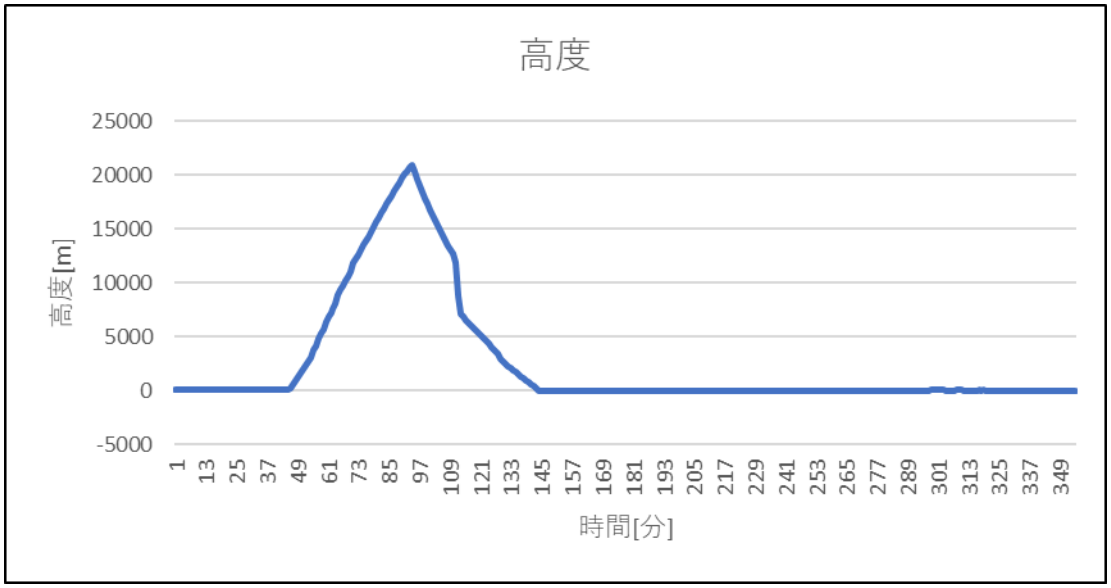


図 27　BMP180 が気圧と海面気圧から自動算出した高度のデータ

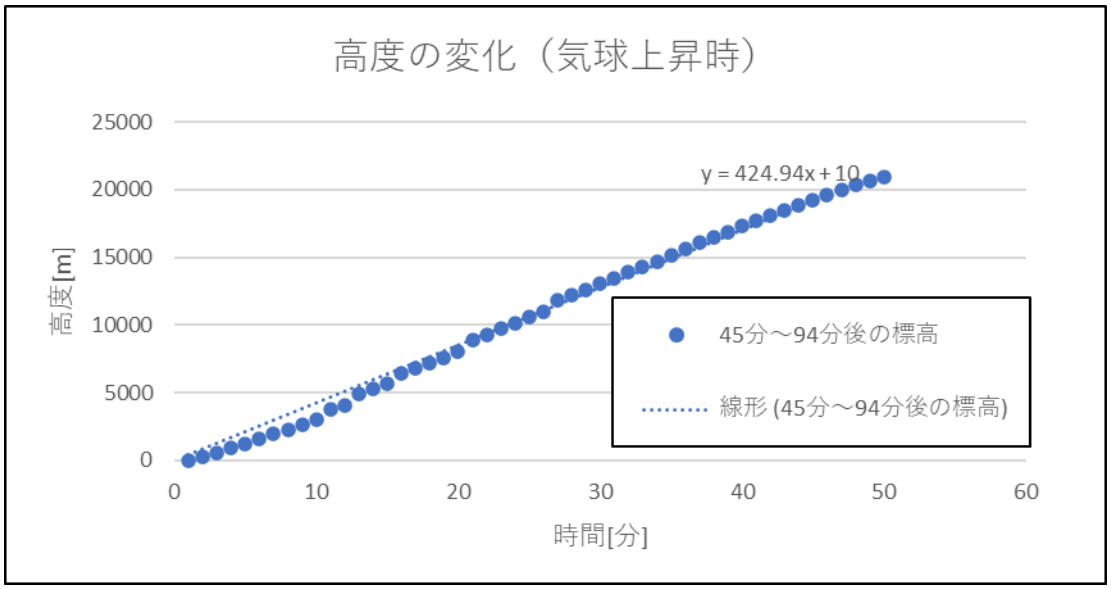


図 28　図 27 における機体上昇時のグラフ

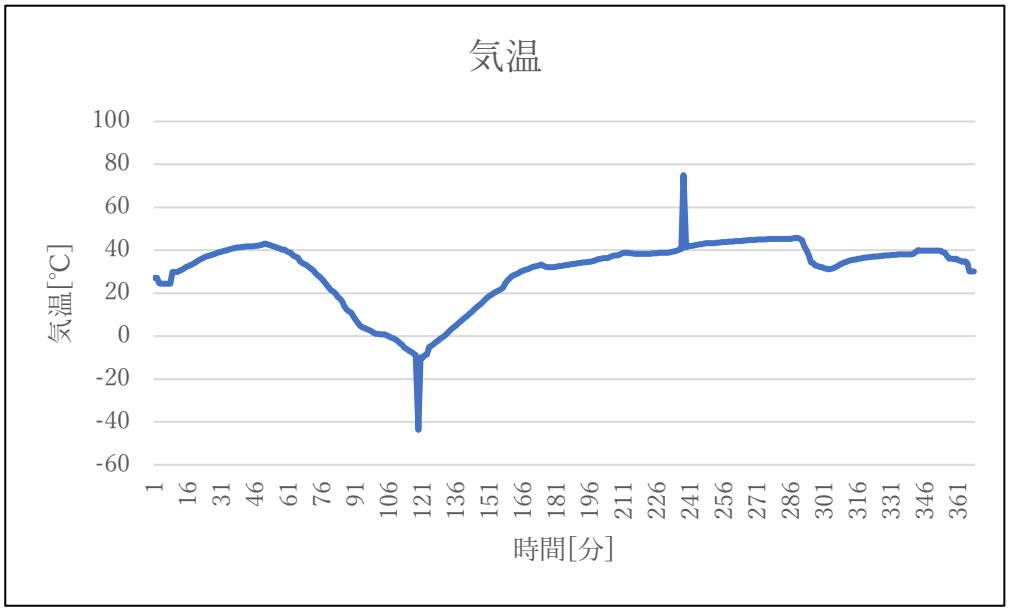


図 29　AHT10 で計測した上空の気温のデータ

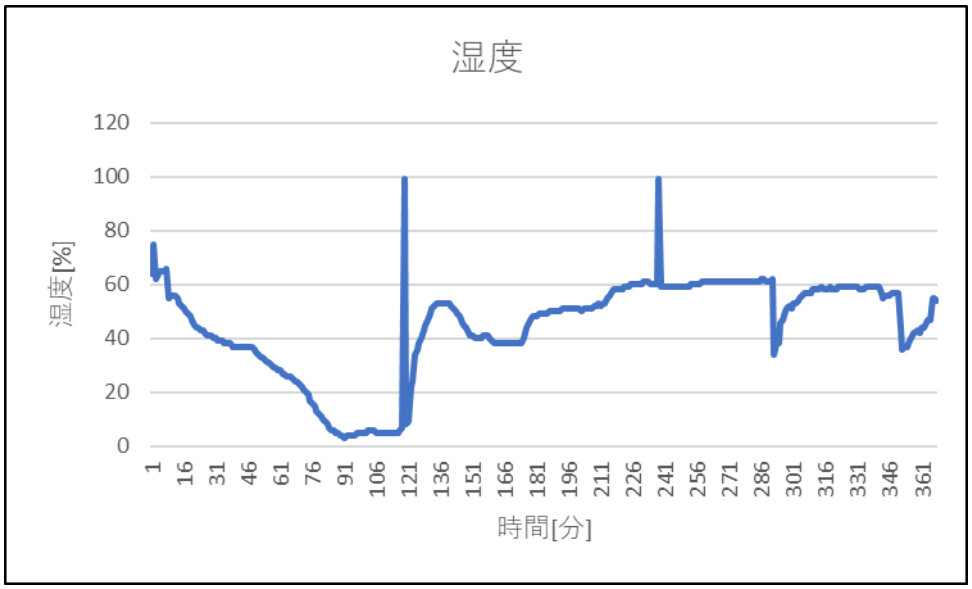


図 30　AHT10 で計測した上空の湿度のデータ

4. 大気回収装置とは別にバルーンサットに備え付けられていた GPS および気圧センサのデータを図 31～33 に示す。

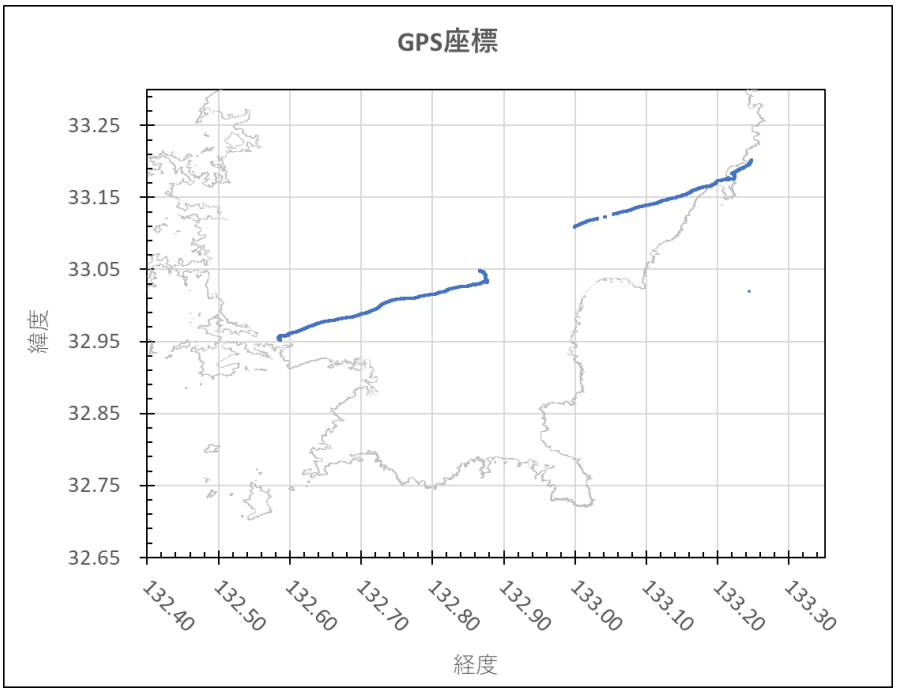


図 31　GPS 座標による飛行経路

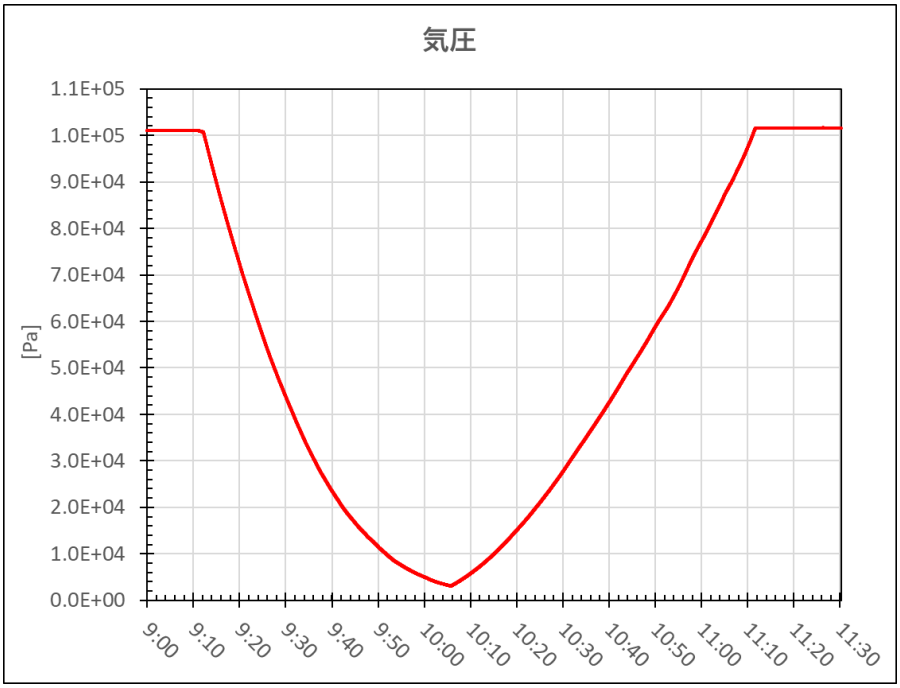


図 32　気圧センサから得られたデータ

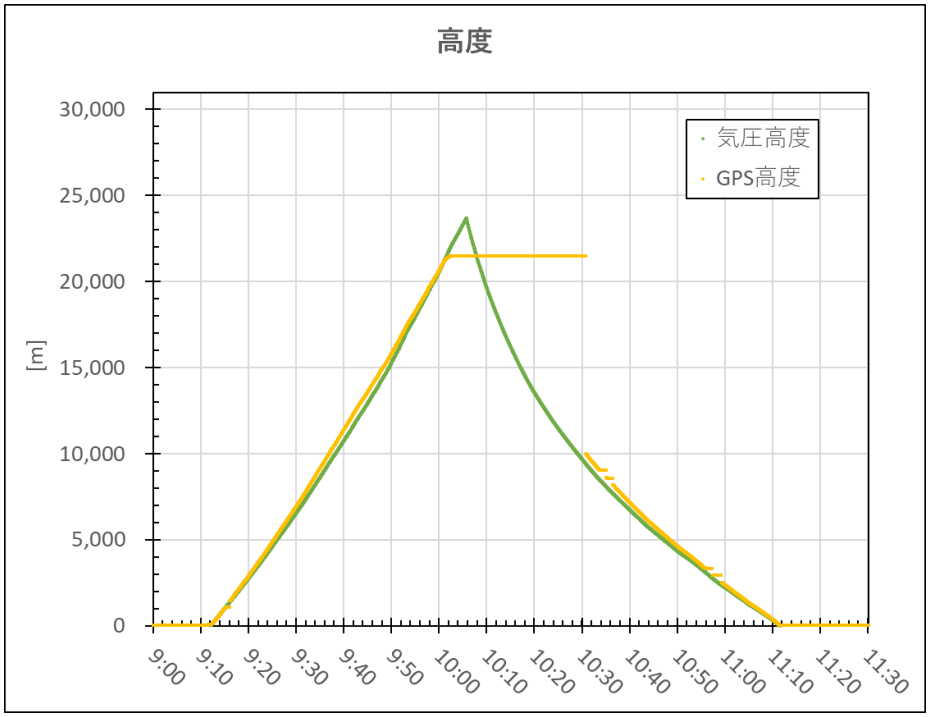


図 33　気圧高度および GPS によって計測した高度

第 6 章　考察

バーティクルメータにより測定したデータの考察をするために粒径による大気物質分類を表 9 に示す。

表 9　粒径による大気物質分類

	考えられる大気中のじん埃粒子	粒径(μm)
1	煙草の煙	0.01~0.9
2	油煙	0.01~0.9
3	海塩粒子	0.05~1.6
4	永久浮遊大気じん埃	0.0009~0.9
5	PM2.5	0.07~2
6	もや	2~30
7	カビ	1.9~17
8	バクテリア	0.25~9
9	沈降性大気じん埃	1~80
10	炭疽菌	0.9~8

空気回収に成功した 2 つの回収袋の内，一方は GPS 制御であるため，正確に 20km 地点の空気を回収できている。また，時間制御で回収した空気は上昇約 50 分後の空気である。よって，この 2 つの回収した空気サンプルを地上の空気サンプルと比較することで, 地上と比較して上空にどれだけ有害な微粒子が含まれているかをさまざまな観点から調査できる。

6-1 地球の大気汚染状況に関する考察

表 10 に前回の大気回収で取得した粒子の数を示す。

表 8 と表 10 を比較すると，今回の計測の方が多く粒子を取得出来ていることが分かる。また，どちらも大気中に含まれている粒子は比較的粒径の小さなものが多いことがわかる。表 6-1 より， 0.07μm～2μm の微粒子のことが PM2.5 と呼ばれており，地上測定できたデータを見ると大気に含まれた粒子のほとんどが PM2.5 に大別できる。このことから，今回の計測において，計測時に中国からやってくる微粒子の数が前回より

も増えていた可能性が高い。

表 10　前回の大気回収で取得したデータ

	高度(km)	粒子数(個)		
		0.3μm	0.5μm	5.0μm
1	0	770	96	1
2	10	1702	395	37
3	20	1406	114	0

6-2 地上と時間制御で取得した大気の比較及び考察

表 8、図、図 21～23 より地上(高度 0km)における粒子数と時間制御で取得した場合における粒子数を比較する。

今回の計測において，粒子数は前回の計測とは逆に，地上の方が多いという結果となった。特に，1.0μm の場合には粒子数に約 2 倍の差がついた。これは，粒子数が風向きや季節などで大幅に変化しやすいことなどから，前回よりも粒子の少ない空域で大気回収を行ったからであると考えられる。

また，前回の計測と同様に粒子径が小さい粒子ほど多く存在していることが分かる。

上記の粒径 0.3μm、0.5μm の微粒子は大気汚染の原因とされている PM2.5 に区分されることから PM2.5 は地上よりも上空に多く含まれていると予想される。

6-3 地上と高度 20㎞ における大気の比較及び考察

表 8、図、図 21～23 より地上(高度 0km)における粒子数と高度 20km における粒子数を比較する。

まず粒径 5.0μm，粒径 1.0μm の粒子数はそれぞれ時間制御で取得した粒子数とほぼ変わっていないことが分かる。しかし，粒径 0.3μm の場合に限り，時間制御で取得した粒子数よりも多くなっていることが分かる。このことから，6-2 の考察を基に考えると，時間制御で回収した大気の高度が，高度 20km よりも高い位置であると考えられる。

6-4 バルーンの経路に関する考察

図 19，20 に実験前のシミュレーション結果，図 31 に実測値を示す。

実験前のシミュレーション結果である図 19，20 と実測値の図 31 を比較すると，経路にはほとんど差が見られないことがわかる。よって，想定通りの経路上の空気を回収できたと考えられる。

6-5 センサのデータに関する解説及び考察

① 気圧及び海面気圧

図 24・図 25 より，マイコンを起動して約 45 分後に気圧が徐々に下がり始めていることが分かる。この値は放球時刻とほぼ一致しており，時刻はほぼ合っていると思われる。このことから，放球時刻 10 時 14 分頃に海面気圧が 3536［Pa］，気圧が 3539［Pa］にまで減少し，最高高度に到達したと思われる。また，11 時 5 分頃に気圧・海面気圧ともに打ち上げ前とほぼ変わらない値まで上昇していることから，沖に着水したと思われる。また，図 31 と気圧を比較すると，機体の下降時に，気圧の減少が歪になっていることが分かる。これは，機体が落下する際に激しく回転したため。計測にばらつきが発生したためであると考えられる。

② 気温及び湿度

図 29 より，ところどころノイズによるバグが含まれているものの，約-10°まで気温が下がっていることが分かる。また，図〇より，湿度はほぼ 0［%］まで減少していることが分かる。また，湿度，気温ともにもとの数値にほぼ戻った後，値が減少している箇所があるが，これは着水後に機体を回収したのちに蓋を開けたためであると思われる。

③ 機体温度

図 30 より，機体温度は約 60 度まで上昇した後，上空の寒気によって冷却され，3 ほどになっていることが分かる。今回実験に使用したマイコンは 0 度までなら耐久が可能であることから，マイコンの耐久性を証明する形となった。

④ 機体高度

図 26 より，約 10 時 24 分頃に，最大高度約 21 ［km］まで気球が到達していることが分かる。また，図 27 より，近似直線の式から，気球は約 425［m/min］の速度で上昇していたと思われる。この速度は，今回使用した気球に想定されている上昇速度である 7［m/s］とほぼ同じ値であることから，センサの値が正確であることがわかる。

また,図 33 のグラフと比較すると,最大高度に約 2 [km]の差があることが分かる。これは，高度を算出した計算式の違いによるものだと考えられる。

6-6 プログラムのバグに関する考察

考えられる原因は，以下の 2 点である。

① GPS が位置情報の取得に時間がかかってしまったことで，位置情報の csv による書き込みが正常に作動しなかった。

② 位置情報の書き込みとエアーポンプの駆動を兼ね，if 文の条件式で位置情報の取得データの書き込みを処理していたことから，if 文の条件式に不備が発生したことで正常に作動しなかった。

第 7 章　まとめ

7-1 まとめ

今回の実験では，地上の大気物質と上空の大気物質を比較すると，前回とは違い，微粒子の数(0.3μm の粒子数)は地上の方が多いという結果となった。このことから，大気の状況はたった一年で大きく変化していることが分かった。しかし，別の視点から見ると汚染物質が地上に降りてきている可能性も考えられる。これら微粒子は、PM2.5 といった大気汚染物質である可能性が高く、もし環境汚染が進んでいる証拠であれば早急な対策が必要である。

また，今回上空の様々な気象データが計測できたことから，大気回収装置としてだけでなく上空の気象観測システムとして動作させることにも成功した。このことから，今後 ICARUS Mark.Ⅱを発展させていくことで，上空の大気汚染を様々な観点から観測できる革新的な気象データベースとして運用できる日もそう遠くないと思われる。

今後も開発を続け，環境汚染に関するさらなる調査を続けていきたい。

7-2 参考文献

- 大気汚染による健康被害は社会問題にも。人体への影響や症状を知ろう (gooddo。jp)
- 環境省_令和元年度　大気汚染状況について（env。go。jp）
- 5-1-11　その他の大気汚染計測器｜JEMIMA　一般社団法人 日本電気計測器工業会
- sensorHAT のライブラリ
- Raspberry Pi で GPS データ取得
- taiki.pdf (cambridgefilter.com)
- ラズパイでモータ制御・TA8428K モータドライバ (101010.fun)
- 気圧センサ bmp180 のデータシート
- 湿度センサ AHT10 のデータシート
- RaspberryPi+AI 電子工作超入門　吉田顕一
- [改訂第 3 版]Linux コマンドポケットリファレンス　沓名亮典