

2019학년도

경기과학고등학교 기초R&E 결과보고서

라즈베리파이를 활용한 자동기상관측장비(AWS)의 제작

2019. 11. 27

한수민(polaris041437@gmail.com)

박서진(tallyfieh1247@gmail.com)

박기현(guitar79@naver.com)

과학영재학교 경기과학고등학교

Contents

Contents	i
List of Figures	ii
List of Tables	iii
초록	iv
I 서론	1
I.1 연구의 필요성	1
I.2 연구의 목적	1
II 이론적 배경	2
II.1 라즈베리파이	2
II.2 라즈비안 (Raspbian)	2
II.3 MariaDB	3
II.4 AWS	3
III 연구 과정	5
III.1 라즈베리파이에 운영체제 설치 및 네트워크 설치	5
III.1.1 라즈비안 설치 및 기본 설정	5
III.1.2 네트워크 및 공유기 설정	5
III.2 센서의 연결	6
III.2.1 온습도 센서(DHT22)	6
IV 결과 및 토의	7
V 결론	10
VI 부록	11
VI.1 AWS Python 코드	11
VI.2 풍속센서 작동을 위한 C언어 코드	14
References	17

List of Figures

Figure 1.	Raspberry pi Logo와 본 연구에서 사용하는 3 B 모델	2
Figure 2.	MariaDB 로고	3
Figure 3.	기상청 AWS 관측 자료와 한국교원대학교에 위치한 AWS 장치	4
Figure 4.	측정한 기상 정보를 DB에 전송한 모습	8
Figure 5.	라즈베리파이 내부 저장소에 저장된 데이터	9

List of Tables

Table 1.	연구에서 사용한 센서의 목록	6
-----------------	---------------------------	---

초 록

본교에는 자동으로 다양한 기상 정보를 수집하는 자동기상관측장비(AWS, Automatic Weather Station)가 있다. 그러나 기존의 자동기상관측장비는 그 가격이 비싸고 학생들이 관측 결과에 접근하기 어렵다는 문제가 있었다. 따라서 본 연구에서는 비교적 가격이 저렴한 라즈베리파이(Raspberry Pi)에 온습도 센서, 기압 센서, 풍향 센서, 풍속 센서, 강우량 센서, 미세먼지 센서로 측정한 값을 Python과 C언어를 통해 저장할 수 있는 자동기상관측장비를 제작하였다. 또한 측정한 정보를 라즈베리파이 내부 저장소에 저장하고 이를 MariaDB의 데이터베이스에 전송 후 저장해 장시간 수집한 데이터를 체계적으로 관리할 수 있도록 하였으며, 추후 센서를 보정하고 수집한 기상 정보를 학교 웹사이트로 전송하여 학생들이 기상 정보를 쉽게 파악할 수 있도록 활용할 수 있을 것이다.

I. 서론

1.1 연구의 필요성

현재 경기과학고등학교에는 자동으로 기상 상황을 관측하는 자동기상관측장비(AWS, Automatic Weather Station)가 하나 존재한다. 그러나 그 가격이 매우 비싸고 AWS의 유지보수가 어려우며 일반 학생들이 수집한 자료에 쉽게 접근하지 못한다는 단점이 존재한다. 또한 온도, 습도, 강우 여부 등의 기본적인 기상 정보는 우리의 일상생활에도 큰 영향을 미치기에 자신이 위치한 지역의 기상 상황을 정확히 필요하는 것이 중요하다. 기상청에서 동네예보를 하고 있으나, 국지적인 기상 상황을 파악하기에는 부족한 점이 존재한다. 그렇기에 학생들이 경기과학고등학교가 위치한 수원시 장안구 송죽동의 기상 상황을 쉽게 확인할 수 있도록 하는 것이 필요하다고 생각되어 본 연구를 진행하게 되었다.

1.2 연구의 목적

본 연구에서는 기존의 AWS 장치가 가지고 있던 단점을 보완하는 것을 목적으로 하였다. 최덕환, 임효혁, 김나영 (2016)의 연구에서는 Mems 센서를 활용하여 소형 AWS를 제작하였으나, 현재까지 라즈베리파이(Raspberry pi)를 사용하여 제작한 자동기상관측장비에 관한 연구는 진행되어 있지 않다. [1] 상대적으로 값이 저렴하고 프로그램의 수정이 쉬운 초소형 컴퓨터 기판인 라즈베리파이와 각종 센서들로 제작하여 AWS의 유지 및 보수를 쉽게 할 수 있도록 하였으며, 수집한 데이터는 라즈베리파이의 내부 저장소에 파일로 저장하고, 동시에 MariaDB 서버 데이터베이스에 전송하였다. 또한 본 연구에서 제작한 AWS 장비를 사용함으로써 경기과학고등학교의 기상 상황을 실시간으로 쉽게 확인할 수 있을 것으로 기대된다.

II. 이론적 배경

2.1 라즈베리파이

라즈베리파이는 영국의 라즈베리파이 재단에서 개발한 초소형 컴퓨터 기판으로 학교 등에서 저가로 컴퓨터 교육을 할 수 있게 하기 위해 개발되었다. 하드 디스크 드라이브(HDD)와 솔리드 스테이트 드라이브(SSD)가 내장되어 있지 않으며 운영체제를 포함한 모든 파일은 마이크로 SD 카드에 저장된다. 본 연구에서 사용된 라즈베리파이의 모델은 Raspberry Pi 3 B이다. 라즈베리 파이 3에는 전원 공급 핀 4개, GND 핀 8개, UART 통신 핀 2개, GPIO 핀 24개 등 총 40개의 핀이 존재하며 이 핀을 통해 각종 센서들을 연결하여 센서에서 측정한 값을 프로그램을 통해 읽어들여 라즈베리파이의 디스플레이에 표시하거나 저장할 수 있다. 아날로그 핀이 존재하는 아두이노와는 달리 라즈베리파이의 GPIO 통신 핀은 모두 디지털 통신 방식을 사용한다는 차이점이 있다. 그 외에 4개의 USB 포트, HDMI 포트 등이 존재하며 전원은 5V의 직류 전압을 사용한다.

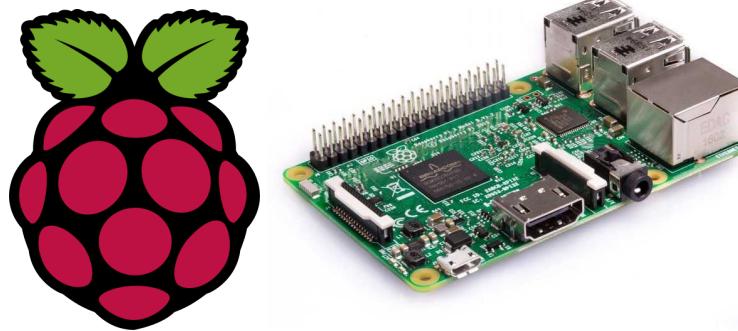


Figure 1. Raspberry pi Logo와 본 연구에서 사용하는 3 B 모델

2.2 라즈비안 (Raspbian)

라즈비안은 라즈베리파이의 운영 체제(OS) 중 하나이며 라즈베리 파이 재단에서 개발한 공식 운영 체제이다. 데비안 리눅스 기반의 운영 체제이며 프로그래밍을 편리하게 하기 위해 Python, Scratch 등의 프로그램을 미리 설치하여 배포하는 Raspbian with Desktop and Recommended Software 버전이 존재하며 연구에서 사용하였다. 2019년 11월 현재 라즈비안의 최신 버전은 4.19이며 본 연구에서는 최신 버전을

사용하였다.

2.3 MariaDB

MariaDB는 오픈소스 관계형 데이터베이스 관리 시스템(RDBMS, Relational Database Management System)이다. 사용하는 소스 코드의 기반이 MySQL과 같기 때문에 Python의 pymysql 모듈과 MySQL에서 데이터를 관리하기 위해 사용하는 코드인 SQL문을 사용해서 MariaDB 서버에 데이터를 전송할 수 있다.



Figure 2. MariaDB 로고

2.4 AWS

AWS는 컴퓨터를 통해 자동으로 기상 정보를 수집하는 장비이다. 기존의 유인 기상관측소에 비해 비용이 적게 들고 정확성이 높다는 장점이 있다. 대한민국 기상청에서는 전국에 500여 개의 AWS를 설치해 온도, 습도, 강수량, 강수 여부, 풍속, 풍향 등의 기상 요소를 수집하고 있으며 수집한 자료는 기상청 날씨누리에 모두 공개되고 있다.



Figure 3. 기상청 AWS 관측 자료와 한국교원대학교에 위치한 AWS 장치

III. 연구 과정

3.1 라즈베리파이에 운영체제 설치 및 네트워크 설치

3.1.1 라즈비안 설치 및 기본 설정

인터넷을 통해 라즈베리파이의 OS인 라즈비안 4.19, 라즈베리파이를 외부 컴퓨터에서 원격으로 접속하기 위한 VNC Viewer, 마이크로 SD 카드에 운영 체제를 전달하기 위한 balena Etcher v1.5.56을 내려 받았다. balena Etcher를 사용해 라즈비안 파일을 마이크로 SD로 전송하고, SD카드는 라즈베리 파이에 삽입하여 라즈베리파이를 구동하였다. 라즈베리파이 2개에 OS를 설치하였으나 실제 센서 연결 및 프로그래밍은 하나의 라즈베리파이에서만 하였다. 구동 후 라즈베리파이에 HDMI선을 이용하여 모니터와 연결한 후 기본 설정을 하였다.

라즈베리파이 장비의 이름은 각각 AWS-01과 AWS-02로 하였고, 외부 접속 비밀번호를 설정하였다. Raspberry pi 3의 경우 WiFi 지역 설정을 한국으로 하면 접속이 되지 않는 오류가 있어 부득이하게 지역을 영국으로 하되 시간은 서울의 표준 시각(KST)을 사용하였다. 라즈베리파이에 GSHS_AWS1 공유기를 연결한 후 라즈베리파이의 내부 IP 주소가 각각 192.168.0.6과 192.168.0.4임을 확인해 노트북의 VNC Viewer를 사용하여 원격 접속하였다. VNC Viewer의 사용을 위해서는 라즈베리 파이 설정에서 VNC를 허용해야 한다.

3.1.2 네트워크 및 공유기 설정

라즈베리파이에서 수집한 정보를 서버로 전송하고 개인 노트북으로 라즈베리파이에 접속해 명령어 작업을 하기 위하여 라즈베리파이를 Wi-Fi 공유기 ‘GSHS_AWS1’에 새로 연결시켰다. 공유기의 IP는 192.168.0.1이다. 그 후 공유기의 DHCP 설정 페이지에서 라즈베리파이의 내부 IP 주소를 AWS-01은 192.168.0.101, AWS-02는 192.168.0.102로 고정시켰다. 라즈베리파이의 네트워크 설정(Network Preferences)에서도 IPv4 주소를 AWS-02 기준으로 192.168.0.102, 라우터 주소를 192.168.0.1, DNS 주소를 168.126.63.1로 설정하였다. 라즈베리파이에 모니터를 연결하지 않고도 외부 컴퓨터에서 작업을 하기 위해서는 컴퓨터에 VNC Viewer를 설치 후 라즈베리파이에 연결해야 한다. 컴퓨터의 네트워크를 GSHS-AWS01에 연결 후 VNC Viewer에서 라즈베리파이에 접속하기 위해 라즈베리파이의 내부 IP 주

소인 192.168.0.102를 입력하였다. 그 후 사용자 이름에 pi를 입력하고 설정한 비밀번호를 입력했을 때 정상적으로 접속이 됨을 확인하였다.

3.2 센서의 연결

라즈베리파이에 각종 기상 정보를 수집할 수 있는 센서를 연결한 후 센서에서 수집한 자료를 처리하는 프로그램을 코딩하였다.

Table 1. 연구에서 사용한 센서의 목록

측정 자료	센서 이름	동작 전압	핀 사용량	비고
온습도	dht22	3.3V	3	10kΩ 저항 사용
기압	BMP180	3.3V	6 (SPI 사용시), 4 (I2C 사용시)	
미세먼지	PMS7003	5V	-	UART 통신 사용, USB 포트 사용
풍향	DM2014	12~24V DC	-	ADC 사용
풍속	SEN0170	7~24V DC	-	ADC 사용
강우량	p/n 80422	5V	2	
ADC	MCP3208	5V	6	풍향, 풍속센서의 연결 위해 필요

3.2.1 온습도 센서(DHT22)

DHT22는 센서가 위치한 지역의 온도와 습도를 측정해주는 센서이다. DHT22는 동작 전압이 5.0 V, 3.3 V 모두 가능하여 3.3 V를 사용하여 아래 회로도와 같이 전선을 연결한다. DHT22 센서의 연결을 위해서는 10

IV. 결과 및 토의

우선 온습도센서는 프로그램을 실행하였을 때 정상적으로 온도와 습도가 표현됨을 알 수 있었으나 기압센서는 SPI 통신을 사용하였을 때 기압이 약 390hPa 정도로 실제와 전혀 맞지 않는 값이 출력되었다. 센서를 교체하여도 같은 현상이 일어났고 센서에서 값을 받아오는 것 자체에는 문제가 없는 것으로 판단되었으므로 센서에서 받아온 값을 기압으로 출력하는 명령어에 문제가 있던 것으로 추정된다. 그러나 SPI 통신은 ADC가 사용하여야 하고 라즈베리파이 하나에는 기본적으로 한 세트의 SPI 통신 판만이 존재하기에 대신 I2C 통신을 사용하여 기압센서를 연결하였으며, 그 결과 기압이 1000mba 내외로 정상적으로 표출되어서 기압 값을 수집할 때 I2C 통신을 사용하기로 하였다.

미세먼지 센서의 경우 프로그램이 실행된 이후 몇 초 동안은 값이 정상적으로 표출되었으나 그 후 원인불명의 오류와 함께 프로그램이 정지하였다. UART 통신에서 값을 받아오는 부분에 문제가 있던 곳으로 추정되며 일반적인 자동기상관측장비에서는 미세먼지 농도를 측정하지 않기에 AWS에 미세먼지 센서는 일단 설치하지 않는 것으로 결정하였다.

ADC의 경우 C언어와 Python으로 컴파일된 프로그램 두 개를 각각 실행한 결과 Python 코드의 경우 오류는 발생하지 않으나 센서의 연결 여부와 무관하게 전압이 계속 0으로 측정되는 문제가 있었다. 반면 C언어 코드의 경우 전압이 정상적으로 출력되었다. AWS를 구동하는 전체 명령어는 Python으로 작성하였으므로 C언어에서 파일 입출력을 통해 텍스트 파일에 수집한 전압 값을 Python에서 다시 한 줄씩 읽어들이는 방법을 선택하였다. 풍속 센서의 경우 라즈베리파이에 (+)극과 GND를 연결하지 않으며 오로지 출력 전압을 연결하는 핀 하나만 ADC에 연결한다.

그렇기 때문에 라즈베리파이에 전원을 공급하는 배터리와 풍속 센서에 전원을 공급하는 배터리가 같아야 두 장치에 연결된 GND의 전위가 같아 값이 정상적으로 표현된다. 1.5V 건전지의 (-)극은 라즈베리파이의 GND에, (+)극은 ADC에 연결하였을 때 약 1350mV 정도의 값이 출력되었으며 건전지의 내부 저항을 고려하면 타당한 값으로 생각된다.

풍속 센서의 출력 전압이 0.4V일 때의 풍속이 0m/s임이 데이터시트에 명시되어 있었으므로 0.4V 전압 이하에서의 풍속을 0으로 가정하였다. 풍향 센서의 경우 12V 전원을 공급한 후 풍향계가 향하는 방향을 달리한 뒤 멀티미터를 사용하여 배터리의 GND와 출력 핀 사이의 전압을 측정한 결과 방향과

무관하게 10V 가량이 나왔다. 데이터시트에 의하면 풍향에 따라 최대 5V의 전압이 출력되어야 하므로 잘못된 값임을 알 수 있다.

강우량 센서의 경우 물방울이 한 번 떨어질 때마다 프로그램을 실행한 이후의 누적 강우량이 정상적으로 출력되었다. 최종 프로그램을 실행하였을 때 온습도와 기압, 단위 시간당 강우량, 풍속이 MariaDB 서버 데이터베이스에 정상적으로 전송됨을 확인하였다.

	보기	구조	SQL	검색	삽입	내보내기	가져오기	테이블 작업	트리거
←→									
□	수정	복사	삭제	21	111111	Suwon	X	1.7	0
□	수정	복사	삭제	22	111111	20191106-105341	X	0	0
□	수정	복사	삭제	23	111111	20191106-105343	X	0	0
□	수정	복사	삭제	24	111111	20191106-105346	X	0	0
□	수정	복사	삭제	28	111111	20191106-110320	X	0	0
□	수정	복사	삭제	29	111111	20191106-110501	X	0	0
□	수정	복사	삭제	30	111111	20191106-110543	X	0	0
□	수정	복사	삭제	31	111111	20191106-110554	X	0	0
□	수정	복사	삭제	32	111111	20191106-110557	X	0	0
□	수정	복사	삭제	37	111111	20191106-111332	X	0	0
□	수정	복사	삭제	38	111111	20191106-111335	X	0	0
□	수정	복사	삭제	39	111111	20191106-111419	X	0	0

Figure 4. 측정한 기상 정보를 DB에 전송한 모습

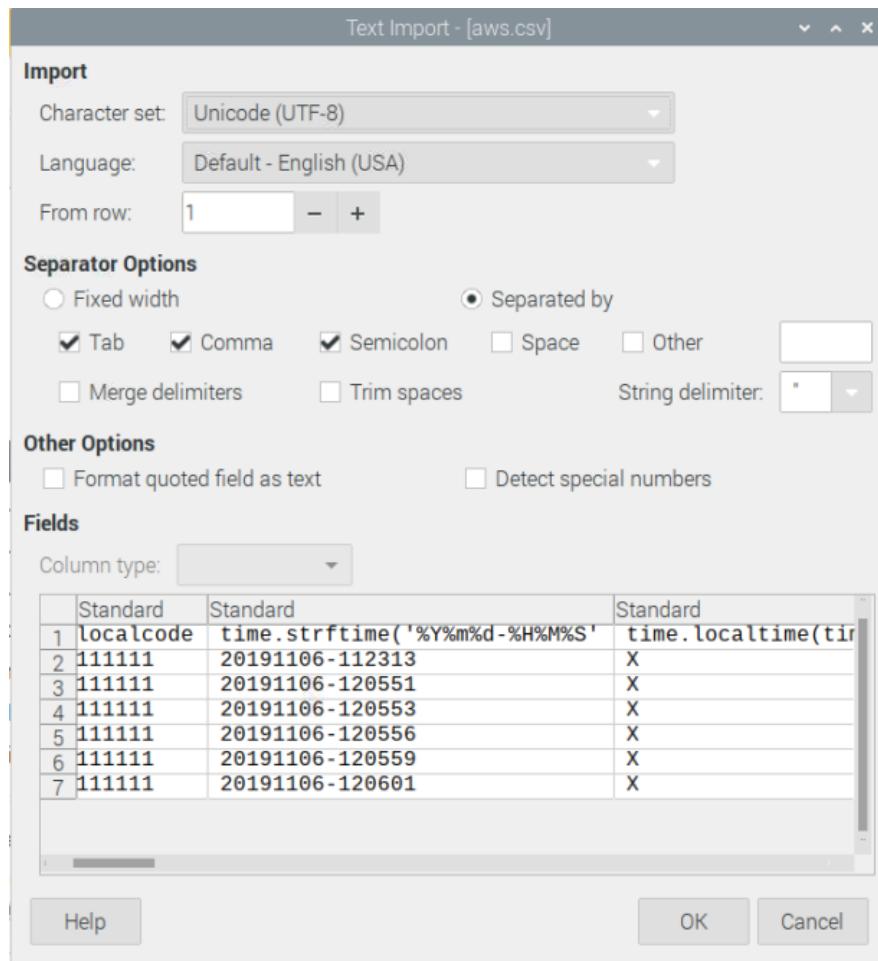


Figure 5. 라즈베리파이 내부 저장소에 저장된 데이터

V. 결론

본 연구에는 라즈베리파이와 각종 센서들을 활용하여 AWS를 제작하고 그 자료를 서버 데이터베이스에 전송하였다. 연구 결과 대부분의 센서가 기상 정보를 정상적으로 수집하였으나 그렇지 않은 센서도 존재하였다.

기압 센서는 통신 방법에 따라 비정상적인 값이 표시되기도 하였으며, 미세먼지 센서의 경우 원인 불명의 이유로 인해 프로그램에 오류가 생겨 자료 수집이 계속 중단되었다.

풍향 센서 역시 풍향계의 방향과 무관하게 전압이 일정하게 표현되어 정보를 수집할 수 없었다. 그렇기에 추후 센서를 라즈베리파이에 연결하는 방법이나 프로그램을 변경해야 할 것으로 보이며 필요시 센서의 종류를 바꾸는 일도 필요해 보인다.

또한 센서에서 표출한 값과 실제 값이 실제로 일치하는지에 대한 확인이 필요하며 만약 차이가 보인다면 보정식을 도출하여 센서에서 수집한 값을 보정해 주어야 할 것이다. 박종서, 오완탁 (1997)의 연구에서는 자동기상관측장비와 백엽상에서 수집한 기온 값의 차이가 $0.1\text{~}0.4^{\circ}\text{C}$ 정도로 측정되었으며, 하지훈, 김용혁, 임효혁, 최덕환, 이용희 (2016)는 회귀분석을 사용해 자동기상관측장비의 기압자료를 보정하는 기법을 제시하였다 [3] [4].

수집한 자료는 데이터베이스에 정상적으로 표출되었으며 이를 통해 장시간 관측으로 인해 생긴 많은 데이터를 체계적으로 관리할 수 있었으며 데이터 유실이나 훼손 등의 문제도 일어나지 않을 것으로 생각된다. 다만 기상청 날씨누리에서 제공하는 AWS 관측 자료와 비교하였을 때 본 장비에서는 수집되지 않는 기상 정보가 더 많았기 때문에 전문적인 기상 관측 자료로는 사용할 수 없으나, 온습도나 해면기압, 현재 강수 여부 등의 간단한 기상 정보는 비교적 쉽게 확인할 수 있다는 점에서 실용성이 있다고 생각된다.

본 연구에서 개발한 AWS는 오류가 발생하거나 센서가 문제가 발생하였을 경우 비교적 쉽게 문제를 해결할 수 있고 센서의 가격 역시 비교적 저렴하기 때문에 AWS 장치 제작 이후의 관리 및 유지보수도 다른 AWS 보다 쉬울 것으로 판단된다. 또한 데이터베이스의 정보를 송죽학사 등의 교내 사이트에 정기적으로 전송해 교내 학생들이 언제나 기상 정보를 쉽고 편리하게 확인할 수 있게 하는 작업이 가능할 것으로 기대된다.

VI. 부록

6.1 AWS Python 코드

```
1 # Raspberry Pi ADC Python Code
2
3 # 0. Modules to import
4 import time
5 import Adafruit_DHT
6 import smbus
7 import pymysql
8 from gpiozero import Button
9
10 rain_sensor = Button(6)
11 BUCKET_SIZE = 0.2794
12 count = 0
13 count15 = 0
14 count60 = 0
15 count3h = 0
16 count6h = 0
17 count12h = 0
18 count1d = 0
19 i = 0
20 now15 = time.time()
21 now60 = time.time()
22 now3h = time.time()
23 now6h = time.time()
24 now12h = time.time()
25 now1d = time.time()
26 rain15 = 0
27 rain60 = 0
28 rain3h = 0
29 rain6h = 0
30 rain12h = 0
31 rain1d = 0
32 localcode = 111111
33 localname = 'Suwon'
34 temp = 0
35 airp = 0
36 rh = 0
37 # 1. Settings
38
39 f = open("windspeed.txt ", 'r');
40 conn = pymysql.connect(host='ess.gs.hs.kr', user='AWS', password='rudrlrhkgkrrh',
41 db='AWS_GSHS', charset='utf8mb4')
42
43 curs = conn.cursor()
44 sql = """ insert into 1min_data(OBS_code, OBS_datetime, preci_now, preci_15, preci_60, preci_3H, preci_6H, preci_12H, preci_1day,
```

```

    temperature , wind_spd01, RH, Air_P, REMARK)
45 values (%s, %s, %s)"""\ # server database information
46
47 while True:
48
49 # 2. Humidity & Temperature
50 sensor_name = Adafruit_DHT.DHT22
51 sensor_pin = 25
52 humidity, temperature = Adafruit_DHT.read_retry(sensor_name, sensor_pin)
53 print ('Temp=_%.2f_C' % temperature)
54 print ('Humid=_%.2f' % humidity)
55 temp = round(temperature, 2)
56 rh = round(humidity, 2)
57 time.sleep(1)
58
59 # 3. Pressure
60 bus = smbus.SMBus(1)
61 bus.write_byte(0x77, 0x1E)
62
63 time.sleep(0.25)
64
65 data = bus.read_i2c_block_data(0x77, 0xA2, 2)
66 C1 = data[0] * 256 + data[1]
67 data = bus.read_i2c_block_data(0x77, 0xA4, 2)
68 C2 = data[0] * 256 + data[1]
69 data = bus.read_i2c_block_data(0x77, 0xA6, 2)
70 C3 = data[0] * 256 + data[1]
71 data = bus.read_i2c_block_data(0x77, 0xA8, 2)
72 C4 = data[0] * 256 + data[1]
73 data = bus.read_i2c_block_data(0x77, 0xAA, 2)
74 C5 = data[0] * 256 + data[1]
75 data = bus.read_i2c_block_data(0x77, 0xAC, 2)
76 C6 = data[0] * 256 + data[1]
77 bus.write_byte(0x77, 0x40)
78
79 time.sleep(0.25)
80
81 value = bus.read_i2c_block_data(0x77, 0x00, 3)
82 D1 = value[0] * 65536 + value[1] * 256 + value[2]
83 bus.write_byte(0x77, 0x50)
84
85 time.sleep(0.25)
86
87 value = bus.read_i2c_block_data(0x77, 0x00, 3)
88 D2 = value[0] * 65536 + value[1] * 256 + value[2]
89 dT = D2 - C5 * 256
90 TEMP = 2000 + dT * C6 / 8388608
91 OFF = C2 * 65536 + (C4 * dT) / 128
92 SENS = C1 * 32768 + (C3 * dT) / 256
93 T2 = 0

```

```

94 OFF2 = 0
95 SENS2 = 0
96 if TEMP >= 2000:
97 T2 = 0
98 OFF2 = 0
99 SENS2 = 0
100 elif TEMP < 2000:
101 T2 = (dT * dT) / 2147483648
102 OFF2 = 5 * ((TEMP - 2000) * (TEMP - 2000)) / 2
103 SENS2 = 5 * ((TEMP - 2000) * (TEMP - 2000)) / 4
104 if TEMP < -1500:
105 OFF2 = OFF2 + 7 * ((TEMP + 1500) * (TEMP + 1500))
106 SENS2 = SENS2 + 11 * ((TEMP + 1500) * (TEMP + 1500)) / 2
107 TEMP = TEMP - T2
108 OFF = OFF - OFF2
109 SENS = SENS - SENS2
110 pressure = (((D1 * SENS) / 2097152) - OFF) / 32768.0) / 100.0
111 cTemp = TEMP / 100.0
112 fTemp = cTemp * 1.8 + 32
113
114 print ("Pressure: %.1f mbar" % pressure)
115 airp = round(pressure, 1)
116 time.sleep(0.25)
117
118
119 # 4. Rainfall
120 def bucket_tipped():
121 global count15
122 global count60
123 global count3h
124 global count6h
125 global count12h
126 global count1d
127 count15 += 1
128 count60 += 1
129 count3h += 1
130 count6h += 1
131 count12h += 1
132 count1d += 1
133
134
135 rain_sensor.when_pressed = bucket_tipped
136
137 if (time.time() - now15 >= 900):
138 rain15 = round(count15 * BUCKET_SIZE, 1)
139 count15 = 0
140 now15 = time.time()
141 if (time.time() - now60 >= 3600):
142 rain60 = round(count60 * BUCKET_SIZE, 1)
143 count60 = 0

```

```

144 now60 = time.time()
145 if (time.time() - now3h >= 10800):
146 rain3h = round(count3h * BUCKET_SIZE, 1)
147 count3h = 0
148 now3h = time.time()
149 if (time.time() - now6h >= 21600):
150 rain6h = round(count6h * BUCKET_SIZE, 1)
151 count6h = 0
152 now6h = time.time()
153 if (time.time() - now12h >= 43200):
154 rain12h = round(count12h * BUCKET_SIZE, 1)
155 count12h = 0
156 now12h = time.time()
157 if (time.time() - now1d >= 86400):
158 rain1d = round(count1d * BUCKET_SIZE, 1)
159 count1d = 0
160 now1d = time.time()
161 if (count15 >= 5):
162 count = 'O'
163 else:
164 count = 'X'
165 print(count)
166
167 # 5. Wind speed
168 line = f.readline()
169 wspeed = line
170 print(wspeed)
171
172 # 6. Time Test
173 print(time.time() - now15)
174
175 # 7. Send info to Database
176 k = time.strftime('%Y%m%d-%H%M%S', time.localtime(time.time()))
177 curs.execute(sql, (
178 localcode, time.strftime('%Y%m%d-%H%M%S', time.localtime(time.time())), count, rain15, rain60, rain3h, rain6h,
179 rain12h, rain1d, temp, wspeed, rh, airp, 'RnE')) # send data to server
180 g = open("aws.csv", 'a')
181 data = """%d, %s, %c, %f, %f, %f, %f, %f, %f, %f, %f, %f\n"""\ %
182 localcode, k, count, rain15, rain60, rain3h, rain6h, rain12h, rain1d, temp, float(wspeed), rh, airp)
183 g.write(data) # save data in raspberry pi
184 g.close()
185 conn.commit()

```

6.2 풍속센서 작동을 위한 C언어 코드

```

1 #include <stdio.h>
2 #include <string.h>
3 #include <errno.h>
4 #include <unistd.h>
5 #include <wiringPi.h>

```

```

6 #include <wiringPiSPI.h>
7
8 #define CS_MCP3208_6 // BCM_GPIO 25
9
10#define SPI_CHANNEL 0
11#define SPI_SPEED 1000000 // 1MHz
12
13
14 int read_mcp3208_adc(unsigned char adcChannel)
15 {
16     unsigned char buff[3];
17     int adcValue = 0;
18
19     buff[0] = 0x06 | ((adcChannel & 0x07) >> 7);
20     buff[1] = ((adcChannel & 0x07) << 6);
21     buff[2] = 0x00;
22
23     digitalWrite(CS_MCP3208, 0); // Low : CS Active
24
25     wiringPiSPIDataRW(SPI_CHANNEL, buff, 3);
26
27     buff[1] = 0x0F & buff[1];
28     adcValue = (buff[1] << 8) | buff[2];
29
30     digitalWrite(CS_MCP3208, 1); // High : CS Inactive
31
32     return adcValue;
33 }
34
35
36 int main (void)
37 {
38     FILE *out;
39     int adcChannel = 0;
40     int adcValue = 0;
41
42     if(wiringPiSetup() == -1)
43     {
44         fprintf (out, "Unable_to_start_wiringPi: %s\n", strerror (errno));
45         return 1 ;
46     }
47
48     if(wiringPiSPISetup(SPI_CHANNEL, SPI_SPEED) == -1)
49     {
50         fprintf (out, "wiringPiSPISetup_Failed: %s\n", strerror (errno));
51         return 1 ;
52     }
53
54     pinMode(CS_MCP3208, OUTPUT);
55     int i = 1;

```

```
56  for(i=1; i<=3600; i++)
57  {
58    out = fopen("windspeed.txt", "a");
59    adcValue = read_mcp3208_adcadc(adcChannel);
60    if(adcValue <= 123) fprintf (out, "0\n"); // too weak voltage
61    else fprintf (out, "% .2f\n", ( float )(20.25*adcValue)/1000)-8.1;
62    fclose (out);
63    sleep(2.54856); // sync with Python code
64  }
65
66
67  return 0;
68 }
```

References

- [1] 최덕환, 임효혁, & 김나영 (2016). Mems 센서를 활용한 소형자동기상관측장비 개발 및 활용방안 연구. *한국기상학회 학술대회 논문집*, 157–158.
- [2] 김용남, 성기홍, 흥정희, & 강동일 (2009). 자동기상관측시스템을 활용한 실시간 기상 관측 자료 제공 웹 페이지 개발. *한국지구과학회지*, 30(4), 478–484.
- [3] 박종서, & 오완탁 (1997). 자동기상관측과 백엽상관측에의한 자료의 비교. *한국기상학회 학술대회 논문집*, 230–233.
- [4] 하지훈, 김용혁, 임효혁, 최덕환, & 이용희 (2016). 소형 자동기상관측장비 (mini-aws) 기압자료 보정 기법. *한국지능시스템학회 논문지*, 26(3), 182–189.