출품번호

제66회 경기도과학전람회

라즈베리파이를 활용한 자동기상관측장비(AWS)의 제작

출품분야	학생	출품부문	지구 및 환경

2020. 6. 17

구 분	성 명
출품자	
지도교사	

Contents

List of Figures

List of Tables

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 사용함로써 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.

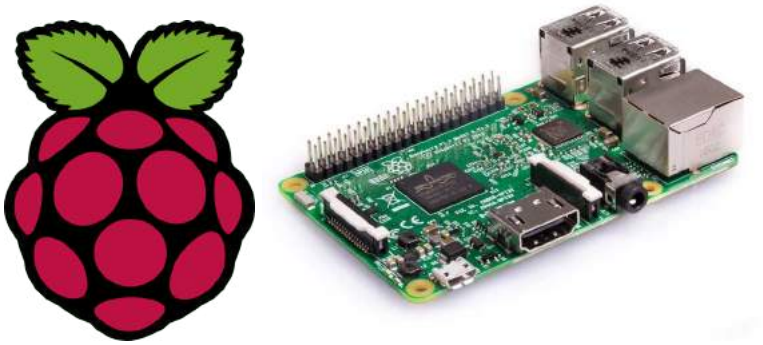

Figure 1. Raspberry pi Logo와 본 연구에서 사용하는 3 B 모델

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.

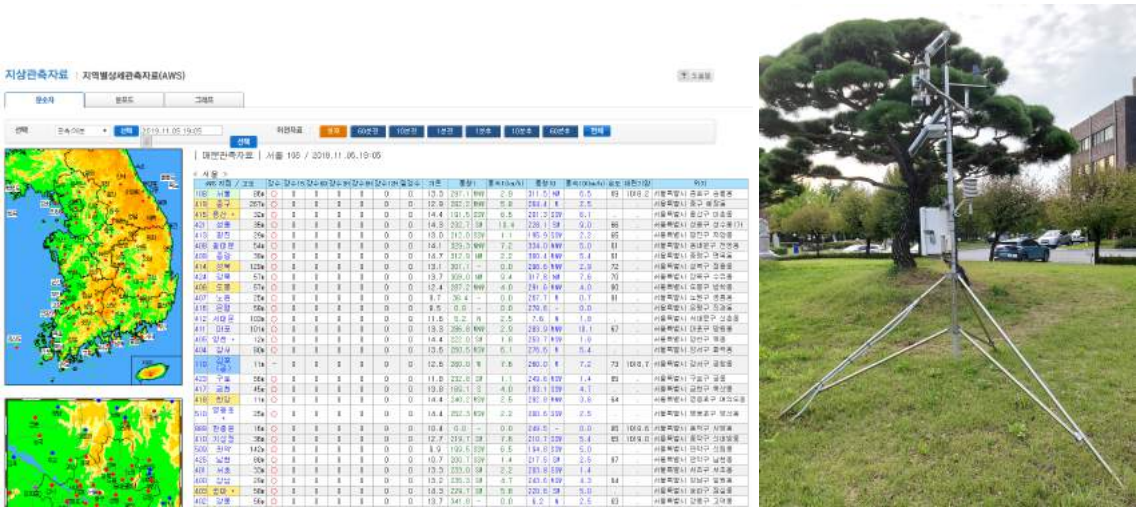

Figure 3. 기상청 AWS 관측 자료와 한국교원대학교에 위치한 AWS 장치

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 내려받았다. 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 Prefernces)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	3.3 V	3	10kΩ 저항 사용
기압	BMP180	3.3 V	6 (SPI 사용시), 4 (I2C 사용시)	
미세먼지	PMS7003	5 V	-	UART 통신 사용, USB 포트 사용
풍향	DM2014	12∼24 V DC	-	ADC 사용
풍속	SEN0170	7∼24 V DC	-	ADC 사용
강우량	p/n 80422	5 V	2	
ADC	MCP3208	5 V	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 kΩ 저항 1개, 3.3 V 전압 핀 1개, GND 핀 1개, GPIO 핀 1개가 필요하다. 전선 연결 후 측정한 온도와 습도를 출력하기 위해 Python 명령어를 실행해 출력한다. 이때 온습도 센서가 값을 받아들여 출력하기 위해서 Python 라이브러리가 필요하므로 Adafruit Python DHT 라이브러리를 다운로드 받아 사용한다.

3.2.2 기압 센서(BMP180)

BMP180은 센서가 위치한 지역의 기압값을 측정하는 센서이며, 온도도 측정 가능하지만 본 연구에서는 이미 온습도 센서가 존재하기에 온도를 측정하는 기능은 사용하지 않았다. BMP180은 3.3 V의 입력 전압을 요구하며 2가지 방법으로 통신이 가능하다. BMP180의 통신에는 SPI(Serial Peripheral Interface Bus) 통신을 사용하는 방법과 I2C(Inter-Integrated Circuit) 통신을 사용하는 방법이 존재하였으며 두 방법

Figure 4. DHT22 센서를 단독으로 연결한 모습

모두를 사용하여 연결을 시도해 보았다.

SPI 통신 사용

아래 배선도와 같이 선을 연결하여 사용해 보았다. 3.3 V 핀 1개, 디지털 출력 핀 4개, GND 핀 1개를 요구하며 Python 코드의 실행을 위해 라이브러리를 라즈베리파이에 설치하였다. 코드는 센서에서 받아들인 값을 기압(hPa 단위)와 해발고도(m 단위)로 환산하여 출력하며 온도는 섭씨온도 단위로 출력한다.

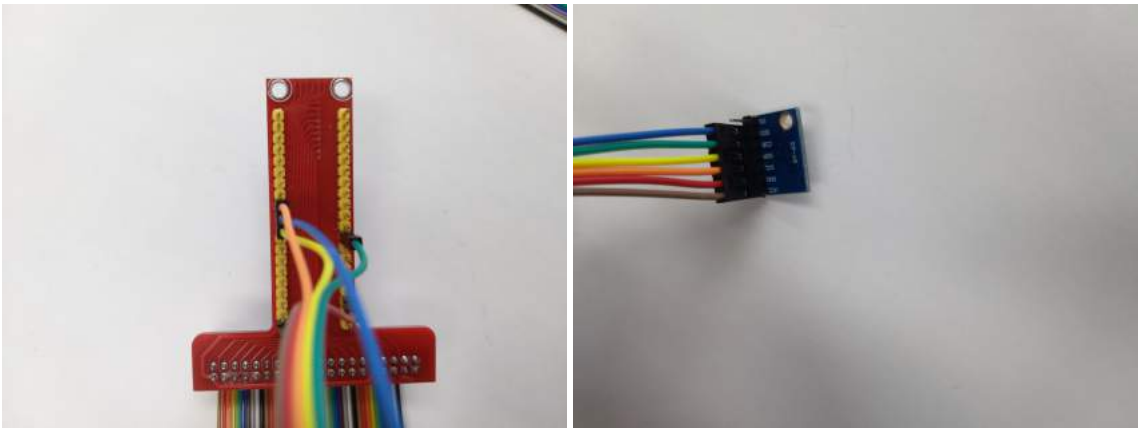

Figure 5. SPI 통신을 사용한 BMP180 센서의 연결

I2C 통신 사용

아래 배선도와 같이 전선을 연결하였다. 3.3 V 핀 1개, GND 핀 1개와 디지털 출력 핀 2개를 사용하였으다. 나머지 핀은 연결하지 않는다. Python 라이브러리를 설치 후 코드를 구동해 센서에서 받아들인 값을 기압(mba 단위)와 온도(섭씨온도 단위)로 출력하였다.

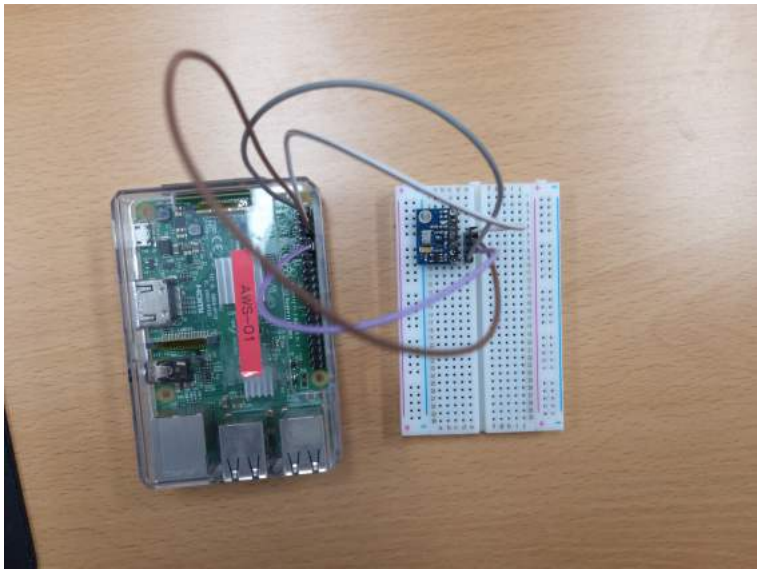

Figure 6. I2C 통신을 사용한 BMP180 센서의 연결

3.2.3 미세먼지 센서(PMS7003)

미세먼지 센서는 공기 중의 미세먼지 농도를 측정하는 센서이다. 미세먼지는 공기 중에 떠다니는 부피 10㎛³ 이하의 작은 먼지 입자를 일컬어 부르는 말이며 크기가 작아 인체에 흡수된 후 쉽게 제거되지 않아 각종 호흡기 질환을 일으키는 원인이기도 하다. 본 미세먼지 센서는 공기 중의 미세먼지에 의해 빛이 산란되는 원리를 이용해 미세먼지의 대기 중 농도를 측정한다. 이때 측정되는 값은 단위 부피당 질량이 아닌 입자수이므로 미세먼지의 평균 질량에 입자수를 곱해 단위부피당 미세먼지의 질량을 구한다.

미세먼지 센서는 UART(Universal asynchronous receiver/transmitter) 통신 방식을 사용하기 때문에 GPIO 핀을 사용하여 연결하지 않으며, USB to TTL을 사용해 센서에 전선을 연결하고 그 전선을 케이블을 통해 라즈베리파이의 USB 포트에 연결하여 통신을 한다. 라즈베리파이 3에서 UART 통신은 기본적으로 비활성화되어 있기 때문에 이를 사용하기 위해서는 환경설정에서 활성화시켜줘야 하며, 이를

위해서는 블루투스 통신을 비활성화시켜야 한다.

미세먼지 센서에 인터페이스 보드를 결합한 후, 아래 그림과 같이 4개의 전선을 연결했다. 이 전선을 USB to TTL 케이블에 연결한 후, 케이블을 라즈베리파이에 연결했다. 그 후 미세먼지서버가 연결된 UART를 확인하고 Python 코드에서 UART명을 입력하는 부분에 현재 센서가 연결된 UART를 입력했다. 결과값 표출 명령어는 PMS7003의 데이터시트와 제공되는 예시 코드를 변형하여 사용하였다.

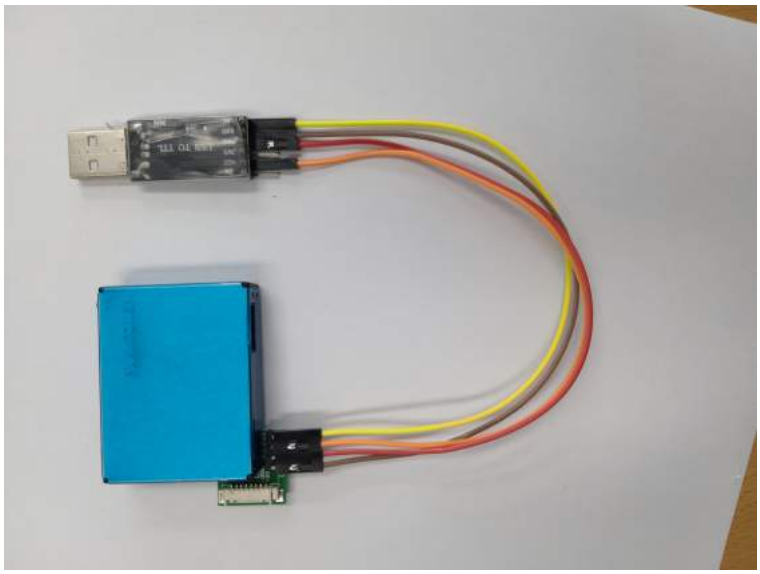

Figure 7. PMS7003 센서를 USB to TTL 케이블을 사용해 연결한 모습

3.2.4 ADC(Analog to Digital Converter, MCP3208)

풍향센서와 풍속센서는 각각 바람의 세기와 방향에 따라 결과를 연속적인 전압 값으로 출력하며 이 값을 읽어들이기 위해서는 아날로그 통신이 필요하다. 그러나 라즈베리파이의 GPIO 핀 중에는 아날로그 통신이 가능한 핀이 없기 때문에 아날로그 신호를 디지털로 바꿔주는 ADC, 즉 아날로그-디지털 변환기를 사용해 값을 받아들인다.

본 연구에서 사용하는 ADC인 MCP3208은 아날로그 입력값을 받아들이는 8개의 핀과 그 값을 라즈베리파이에 전송하기 위해 사용하는 8개의 핀으로 구성되어 있다. ADC를 라즈베리파이에 다음 그림과 같이 연결했으며 C언어와 Python을 각각 사용해 구동시켜 보았다.

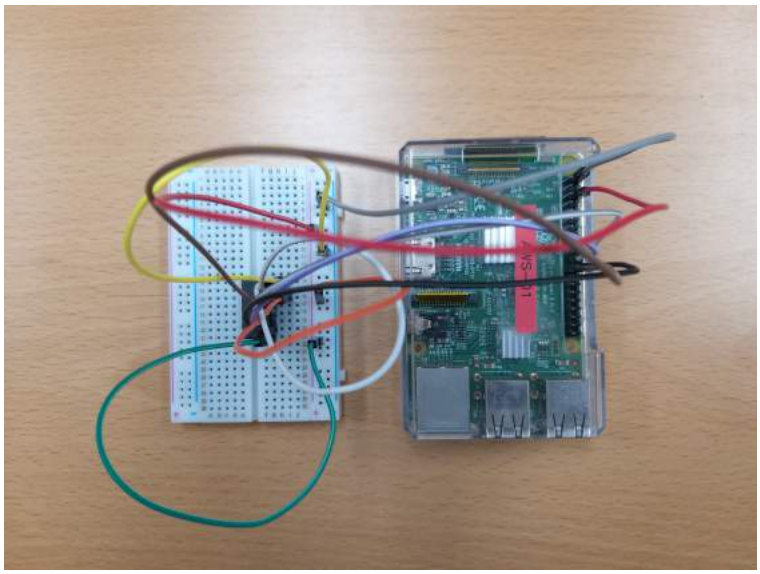

Figure 8. ADC(MCP3208)을 라즈베리파이에 연결한 모습

3.2.5 풍속센서(SEN0170)

풍속센서는 바람이 흐르는 속력을 측정하는 센서이다. 풍속센서의 풍속계는 바람이 세게 불수록 더 빨리 회전하며, 이 회전하는 정도를 연속적인 전압값으로 출력한다. SEN0170의 데이터시트에는 풍속이 0 m/s일 때 출력 전압이 0.4 V, 풍속이 32.4 m/s 때의 출력 전압이 2.0 V로 표시되어 있었다. 32.4 m/s는 본 센서가 측정할 수 있는 최대 풍속이며 그러므로 출력되는 최대 전압은 2.0 V를 넘지 않는다.

본 연구에서는 전압과 풍속의 상관관계가 선형적이라고 가정한 뒤 전압에 따른 풍속을 구하는 공식을 도출하였다. 전압을 V [V], 풍속을 v [m/s]라 할 때 둘 사이에는

$$v = 20.25V - 8.1 \quad (V \geq 0.4\text{V})$$

의 관계가 성립함을 알 수 있었다. 이때 전압이 0.4 V 미만이라면 풍속은 0 m/s로 표시하도록 하였다. 풍속센서에는 4개의 전선이 존재한다. 2개는 각각 (+)와 GND를 연결하는 전선이며, 1개는 전압을 출력하는 전선이고, 나머지 1개의 전선은 사용하지 않는다.

데이터시트에 의하면 풍속센서는 7~24 V의 직류 전압 사이에서 동작하므로 12 V 배터리에 센서를 연결하여 사용하기로 하였다. (+)극 선과 GND 선을 12 V 배터리에 연결할 수 있도록 케이블과 납땜하였으며, 풍속센서의 출력 전선을 ADC의 CH0(아날로그 전압값을 읽어들이는 핀)에 연결한 후 명령어를

작성하고 실행해 풍속이 올바르게 표출되는지 확인하였다. 명령어는 ADC에서 사용한 코드에서 읽어들인 전압값을 이용해 풍속을 계산하여 출력하도록 하였다.

3.2.6 풍향센서(DM2014)

풍향센서는 바람이 부는 방향을 출력하는 센서이다. 데이터시트에 의거하면 풍향센서는 바람이 부는 방향에 따라 16방위로 나누어 각 방위에 따라 전압값을 다르게 표현하며, 이를 이용해 바람이 불어오는 방향을 알 수 있다.

풍향센서에는 4개의 핀이 존재하며 2개는 각각 (+)극과 GND를 연결하는 선이고 1개는 전압을 출력하는 전선, 나머지 1개는 사용하지 않는 전선이다. 풍향센서의 동작전압은 12~24 V 사이의 직류 전압이므로 (+)극 선과 GND 선이 배터리에 연결될 수 있도록 전선을 케이블에 납땜한 뒤 12 V의 배터리에 풍향센서를 연결하였다. 명령어를 작성하고 구동하기 전에 멀티미터를 사용하여 바람의 방향이 변함에 따라 전압값이 제대로 출력하는지 확인하였다. 멀티미터를 사용하여 GND 핀과 출력 전압 핀 사이의 전압을 측정하였다.

Figure 9. 풍속센서와 풍향센서의 모습

3.2.7 강우량센서(p/n 80422)

강우량센서는 빗방울이 떨어질 때만 전기 신호가 전달되는 방식으로 강우량을 측정하는 센서이다. 강우량센서에 빗방울이 떨어지면 강우량센서의 하부에 위치한 시소 모양의 건축물이 기울어지게 된다. 이에 따라 버튼이 눌려지고, 버튼이 한 번 눌려질 때마다 신호가 전달되어서 버튼이 눌린 횟수에 따라 내린 비의 양을 알 수 있게 된다.

센서는 강우량을 비 한 방울의 부피와 버튼이 눌러진 횟수를 곱해 구한다. 강우량 센서의 전선을 다음과 같이 연결한 후 명령어를 작성해 실행 여부를 확인했다. 강우량 센서에 빗방울이 떨어진 경우에만 프로그램이 값을 출력하게 되어 있어 비가 내리지 않는 경우 아무런 값도 출력하지 않는다.

Figure 10. 강우량센서를 라즈베리파이에 연결한 모습

3.3 AWS 구조물의 보수와 센서의 기판 연결

본교의 SRC(Science Research Center) 7층 야외에 위치해 있으나 현재 사용하지 않는 AWS 구조물을 보수하고 센서와 전선을 연결했다. 과거 AWS 구조물은 알루미늄 프로파일을 이용해 제작되였으며 양쪽 벽면에 와이어를 이용해 고정시킨 상태였으나 붕괴된 적이 있어 추가적인 보수가 필요한 상황이었다. 본 연구에서는 알루미늄 프로파일의 접합부에 볼트와 너트를 2중으로 설치하였으며, 구조물의 양쪽을 와이어를 사용해 다시 벽에 단단하게 고정하였다. 또한 구조물의 상단부에 풍속 센서와 풍향 센서를 설치하고

전선을 케이블 타이로 알루미늄 프로파일에 고정하였다. 구조물의 측면에는 태양광 전지판을 설치했다.

구조물의 중간 부분에는 라즈베리 파이와 배터리, 풍향, 풍속, 강수량을 제외한 각종 센서가 위치할 공간이 위치해 있다. 상자에 라즈베리 파이와 온습도, ADC, 기압 센서를 연결한 기판을 놓은 후 상자에 펀치를 이용해 전선이 들어갈 구멍을 뚫었다. 그 후 배터리와 상자를 구조물 위에 놓고 풍속, 풍향, 강우량 센서의 전선과 배터리의 전선을 상자 안에 넣어 라즈베리파이와 연결했다.

풍향과 미세먼지 센서를 제외한 모든 센서의 전선과 ADC, 라즈베리파이 확장 보드(GPIO Extension Board)를 만능기판에서 서로 납땜하여 연결했다. 라즈베리파이 확장 보드는 라즈베리파이에 연결하였을 경우 라즈베리파이에 기본적으로 내장되어 있는 핀과 동일한 용도로 사용할 수 있으며 전선의 연결과 납땜, 이후 유지 보수를 쉽게 해주는 역할을 한다.

3.4 서버 데이터베이스로 수집한 정보 전송

김용남, 성기훈, 홍정희, 강동일 (2009)은 자동기상관측장비에서 수집한 데이터를 표출하는 웹 페이지를 개발하는 연구를 진행하였으나 이는 미리 설치되어 있던 AWS에서 값을 받아와 웹 페이지에 표출하는 형태였다 [2].

본 연구에서는 MariaDB를 통해 라즈베리파이를 이용하여 직접 수집한 기상 정보를 체계적으로 정리할 데이터베이스를 구축한 후 이를 웹상에서 관리하기 위해 경기과학고 지구과학 홈페이지에 설치된 phpMyAdmin(ess.gs.hs.kr/phpmyadmin)을 이용하였다. 서버에 기상 정보를 저장하는 DB인 AWS_GSHS를 만든 후 아래와 같이 테이블을 제작하였다. 테이블의 이름은 1min_data로 하였으며 내용은 기상청 날씨누리에서 제공하는 AWS 관측자료의 형식을 따랐다. 단 본 연구에서 제작한 AWS 장비에서 현재 측정할 수 없는 자료는 NULL값을 기본으로 저장하기로 설정하였다.

라즈베리파이에서 수집한 값을 서버에 전송하기 위해 Python에서 pymysql 모듈을 불러온 후 SQL 문을 사용하였다. C언어와 Python 명령어에서 반복문을 실행하는 시간 간격이 같도록 프로그램의 시간 딜레이를 설정하였다. 또한 수집한 정보는 라즈베리파이 내부에도 csv(comma separated value) 파일의 형태로 저장하도록 설정하였다.

Table 2. AWS_GSHS DB의 1min_data 테이블 구조

이름	설명	데이터형	기본값	비고
id	데이터 일련번호	int		자동으로 부여
OBS_code	AWS 지역코드	int		임의로 111111 부여
OBS_datetime	측정시각	varchar		연월일-시분초로 표시
preci_now	현재 강수여부	varchar		O, X로 표시
preci_15	15분 누적강우량	float		
preci_60	60분 누적강우량	float		
preci_3H	3시간 누적강우량	float		
preci_6H	6시간 누적강우량	float		
preci_12H	12시간 누적강우량	float		
preci_1day	1일 누적강우량	float		
temperature	기온	float		
wind_dir011	1시간 풍향(360도)	float	NULL	
wind_dir012	1시간 풍향(16방위)	float	NULL	
wind_spd01	1시간 풍속	float		
wind_dir101	10시간 풍향(360도)	float	NULL	
wind_dir102	10시간 풍향(16방위)	float	NULL	
wind_spd10	10시간 풍속	float	NULL	
RH	상대습도	float		
Air_P	대기압	float		
REMARK	비고	varchar		‘RnE’로 표시

IV. 결과 및 토의

온습도센서의 경우 프로그램을 실행하였을 때 정상적으로 온도와 습도가 표현되었다. 그러나 기압센서는 SPI 통신을 사용하였을 경우에는 약 390hPa 정도의, 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 명령어, 또는 Python 라이브러리에 문제가 있던 것으로 추정된다. 그 외에도 SPI 통신은 ADC가 사용하여야 하고 라즈베리파이 하나에는 기본적으로 한 세트의 SPI 통신 판만이 존재하기에 I2C 통신을 사용하여 기압센서를 연결하기로 하였으며, 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 있거나, 라즈베리파이에 의해 공급하는 전력이 부족해서 생기는 문제로 추정되며, 일반적인 자동기상관측장비에서는 미세먼지 농도를 측정하지 않기 때문에 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 전압 값을 라즈베리파이에 저장하고, 그 저장한 값을 Python에서 다시 한 줄씩 읽어들이는 방법을 선택하였다. 풍속 센서의 경우 라즈베리파이에 (+)극과 GND를 연결하지 않으며 오로지 출력 전압을 연결하는 핀 하나만 ADC에 연결하였다.

그렇기 때문에 라즈베리파이에 전원을 공급하는 배터리와 풍속 센서에 전원을 공급하는 배터리가 같아야 두 장치에 연결된 GND의 전위가 같아 값이 정상적으로 표현된다. 측정되는 전압값이 합리적인 수치인지 시험해 보기 위해 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 0.4 V일 때의 풍속이 0 m/s임이 데이터시트에 명시되어 있었으므로 0.4 V 전압 이하에서의 풍속을 0으로 가정하였다. 풍향 센서의 경우 12 V 전원을 공급한 후 풍향계가 향하는

방향을 달리한 뒤 멀티미터를 사용하여 배터리의 GND와 출력 핀 사이의 전압을 측정한 결과 방향과 무관하게 10 V 가량의 전압이 출력되었다. 데이터시트에 의하면 풍향에 따라 최대 5 V의 전압이 출력되어야 하므로 잘못된 값임을 알 수 있으나 값이 잘못 출력된 이유는 찾지 못했다.

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

서버: localhost:3306 » 데이터베이스: AWS_GSHS » 테이블: 1min_data

보기 | 구조 | SQL | 검색 | 삽입 | 내보내기 | 가져오기 | 테이블 작업 | 트리거

	id	OBS_code	OBS_datetime	preci_now	preci_15	preci_60	preci_3H	preci_6H	preci_12H	preci_1day	temperature	wind_dir011	wind_dir012
수정 복사 삭제	21	111111	Suwon	X	1.7	0	0	0	0	0	22.9	NULL	NULL
수정 복사 삭제	22	111111	20191106-105341	X	0	0	0	0	0	0	22.1	NULL	NULL
수정 복사 삭제	23	111111	20191106-105343	X	0	0	0	0	0	0	22	NULL	NULL
수정 복사 삭제	24	111111	20191106-105346	X	0	0	0	0	0	0	22	NULL	NULL
수정 복사 삭제	28	111111	20191106-110320	X	0	0	0	0	0	0	21.8	NULL	NULL
수정 복사 삭제	29	111111	20191106-110501	X	0	0	0	0	0	0	21.8	NULL	NULL
수정 복사 삭제	30	111111	20191106-110543	X	0	0	0	0	0	0	21.8	NULL	NULL
수정 복사 삭제	31	111111	20191106-110554	X	0	0	0	0	0	0	21.8	NULL	NULL
수정 복사 삭제	32	111111	20191106-110557	X	0	0	0	0	0	0	21.8	NULL	NULL
수정 복사 삭제	37	111111	20191106-111332	X	0	0	0	0	0	0	21.8	NULL	NULL
수정 복사 삭제	38	111111	20191106-111335	X	0	0	0	0	0	0	21.7	NULL	NULL
수정 복사 삭제	39	111111	20191106-111419	X	0	0	0	0	0	0	21.7	NULL	NULL

Figure 11. 측정한 기상 정보를 DB에 전송한 모습

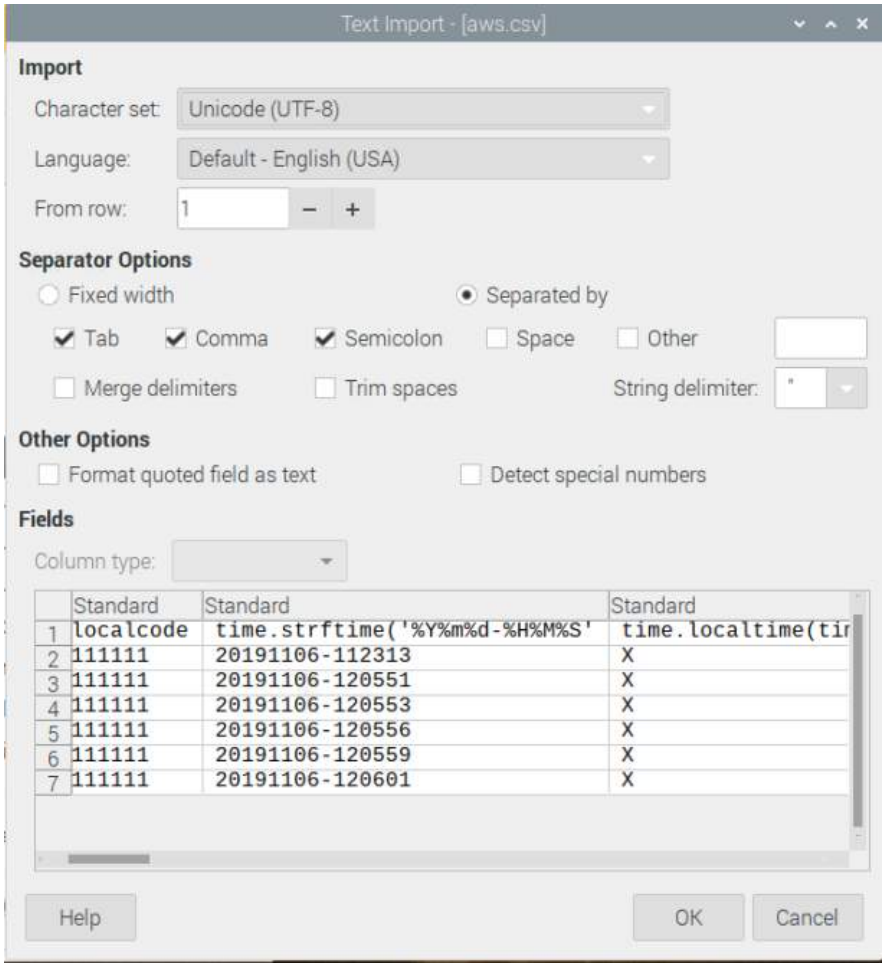

Figure 12. 라즈베리파이 내부 저장소에 저장된 데이터

V. 결론

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

기압 센서는 SPI 통신을 사용할 경우 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 센서 자체가 불량일 경우를 대비해 센서의 종류를 바꾸는 일도 필요해 보인다.

또한 센서에서 표출한 값과 실제 값이 실제로 일치하는지에 대한 확인이 필요하며 만약 차이가 보인다면 센서에서 수집한 값을 보정하는 작업이 필요할 것이다. 박종서, 오완탁 (1997)의 연구에서는 자동기상관측장비와 백엽상에서 수집한 기온 값의 차이가 0.1˜0.4°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 기기에 오류가 발생하거나 센서에 문제가 생겼을 경우 비교적 쉽게 문제를 해결할 수 있고, 센서의 가격 역시 10만 원을 넘지 않아 비교적 저렴하기 때문에 AWS 장치 제작 이후의 관리 및 유지보수도 다른 AWS 보다 쉬울 것으로 판단된다. 또한 데이터베이스에 저장된 정보를 교내 사이트에 정기적으로 전송하는 작업을 거친다면 교내의 학생들이 학교의 기상 상황을 쉽고 편리하게 확인할 수 있을 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, %s, %s, %s, %s, %s, %s, %s, %s, %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_adc(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, 홍정희, & 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.