RX-9 Application Note

EXSEN Inc.

Model Name	**RX-9, RX-9 Simple**
Date	2020.03.25
Remark	R0

Revision History

Rev No	Description	Date	Note
R0	Initiate Document	20.03.25	ykkim

본 문서는 RX-9, RX-9 Simple 구동 원리와 샘플 코드에 대해서 설명합니다.

목차

1 구동 원리

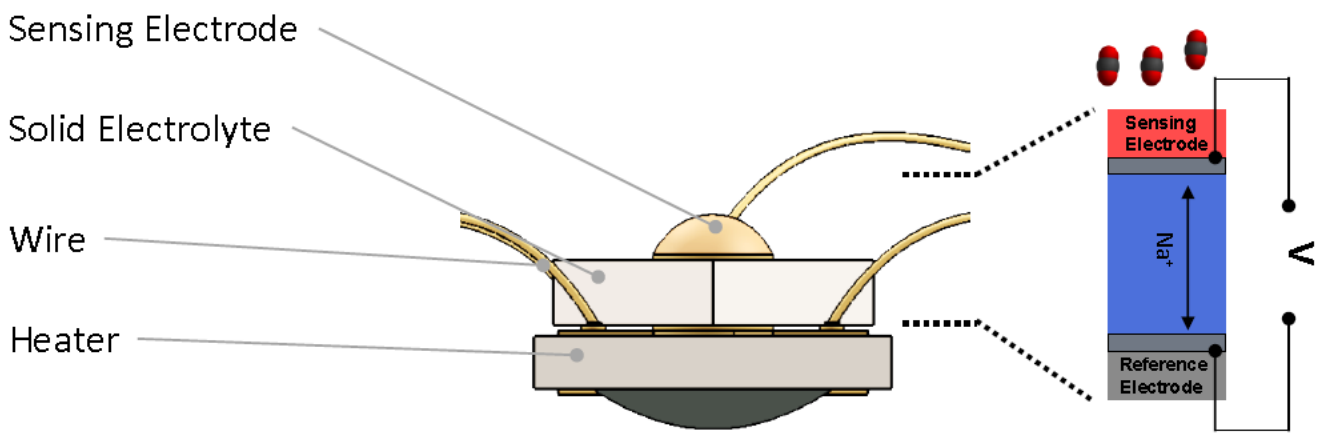


Figure 1 Internal Structure of sensor package

RX-9 은 고체전해질 전기 화학식 센서로써 감지전극(Sensing Electrode) 주변의 이산화탄소 농도에 따라 고체전해질 내부의 Na^+ 이온의 이동도가 변화되어 발생하는 전위차로 이산화탄소 농도를 유추하는 방식입니다.

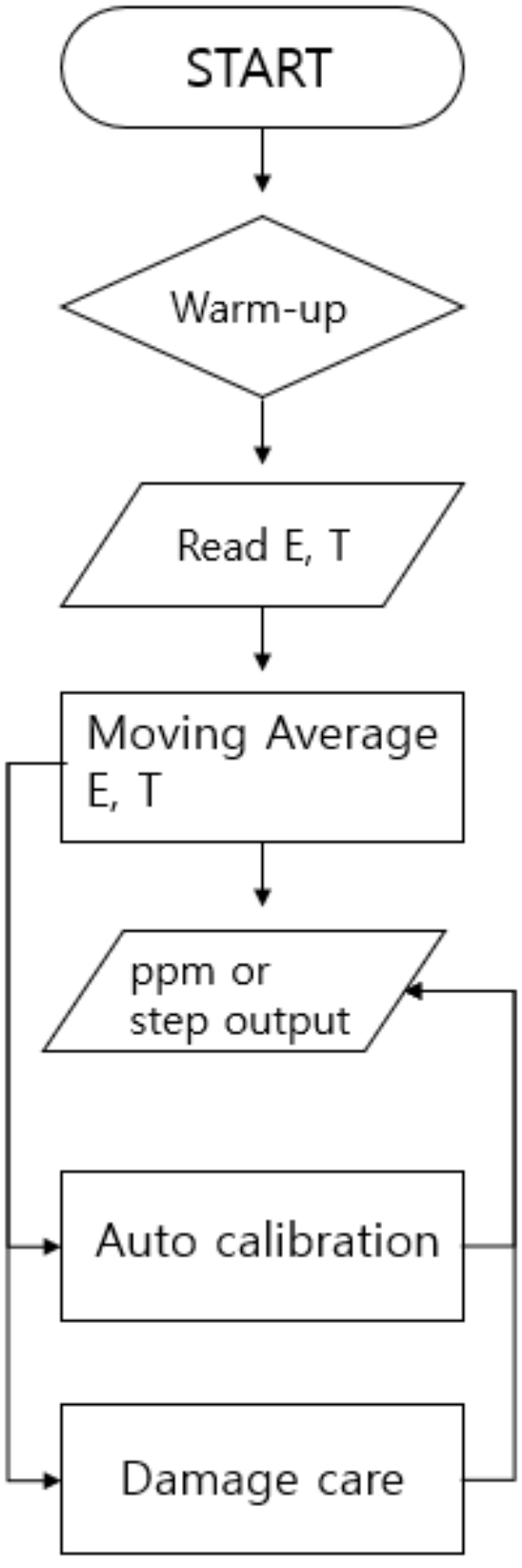
고체전해질 내부의 이온의 이동을 원활하게 하기 위해 고체전해질을 고온으로 가열할 필요가 있으며, 그 온도는 섭씨 350~450 도 정도입니다. 비가동시에는 고체전해질이 완전히 식어 있는 상태이기 때문에 사용하기 위해서 다시 가열하는 데 약 3 분의 시간이 소요가 됩니다. 이 시간을 warm-up 시간이라고 하며, 센서가 완전히 가열되지 않은 상황에서 측정되는 센서의 신호(EMF)는 사용할 수 없습니다.

센서의 온도에 따라 센서의 신호가 달라질 수 있습니다. 이 때문에 RX-9 은 Thermistor 를 센서 주변에 설치하여 센서의 온도를 읽도록 설계되었습니다. RX-9 의 T 핀이 그 역할을 수행하며 온도 보상용으로 사용이 됩니다.

센서의 신호(EMF)는 Electromotive Force 라고 하며 전압의 형태로 측정이 됩니다. 보통 100~500 mV 사이의 값이지만, 너무 낮은 수치이기 때문에 OPAMP 로 증폭하여 E 핀을 통해서 출력합니다. RX-9 의 경우 6 배 증폭하여 출력이 됩니다.

E 핀과 T 핀에 입력된 신호를 ppm 계산 수식에 넣어서 실시간으로 계산하게 됩니다. RX-9 Simple 과 RX-9 은 동일한 계산 수식을 사용하지만, RX-9 은 교정 데이터를 1:1 로 매칭한 상태에서 계산하지만, RX-9 Simple 은 대표값으로 계산한다는 점이 다릅니다. RX-9 Simple 은 1:1 교정데이터가 없기 때문에 ppm 수치로 출력하는 것은 권장하지 않으며, 단계 출력용으로만 사용하기를 권장합니다.

2 순서도



START: 센서 구동 시작, 전원 인가

Warm-up: 센서 구동 후 3 분 이내에는 정확한 수치를 출력할 수 없음. Warm-up 시간 내에는 센서 수치를 표시하지 않음

Read E, T: 센서의 E 핀과 T 핀으로부터 아날로그 신호를 수신

Moving Average E, T: 아날로그 신호의 흔들림 때문에 이동평균을 내서 사용함

Ppm or step output: 센서의 구동 알고리즘에 따라 E 핀과 T 핀에서 읽은 값으로 이산화탄소 농도를 계산함

Auto calibration: 센서에서 출력되는 ppm 값들 중 최소값을 지구 대기 농도에 맞추는 알고리즘. 센서가 구동 중에 값이 틀어지는 것을 보정하기 위해서 주기적으로 센서의 값을 자동으로 보정함

Damage care: 센서의 값이 비정상적으로 영향을 받은 경우, 센서의 값을 빠른 시간 내에 바로잡기 위한 알고리즘

3 코드 설명

기본적인 샘플코드는 https://github.com/EXSEN/RX-9 의 RX-9_SAMPLE_CODE_WO_EEP를 참고하시면 됩니다. 코드는 github에 최신 버전을 유지합니다.

```
/*
   coded by EXSEN
   date: 2020.03.27
   CO2 sensor is attached to ATMEGA328P, 16 Mhz, 5V
   Board Ver: not specified
   file name: RX-9_SAMPLE_CODE_WO_EEP_200303.ino
   you can ask about this to ykkim@exsen.co.kr
   CAUTIONS
   1. Don't use 3.3V of arduino to RX-9 V, RX-9 consume more than arduino's 3.3V output.
   2. Use external 3.3V source. you can use this
(https://ko.aliexpress.com/item/1996291344.html?spm=a2g0s.9042311.0.0.27424c4dytyPzF)
   All remarks are written under the code to explain
   모든 주석은 설명하고자 하는 코드 아래에 쓰여집니다.
*/
const String VER = "RX-9_SAMPLE_CODE_WO_EEP_200312";
#define EMF_pin 0  //RX-9 E, Analog, A0 of Arduino, 아날로그 핀 연결 번호
#define THER_pin 1 //RX-9 T, Analog, A1 of Arduino, 아날로그 핀 연결 번호
#define Base_line 432
/* 지구의 최저 이산화탄소 농도, 국제 표준 이산화탄소 농도인 하와이 측정값은 2020년에 413.4 ppm 이었음
   하지만, 하와이는 매우 청정한 구역이기 때문에 우리가 지내는 장소와는 최저 농도가 차이가 있음
   이를 보정하기 위해 본 센서의 최저농도를 하와이의 농도보다는 약간 더 높게 설정할 필요가 있음
   보정 수치
      - 대도시, 주변에 교통량이 많은 경우: +40~80 ppm
      - 대도시, 보통 통행량: +20~40 ppm
      - 교외 지역: + 10~20 ppm
      - 도로로부터 10분이상 떨어진 곳, 차량이 근접할 수 없는 곳: + 0~10 ppm
   일반적으로 가솔린 엔진 기관에서 이산화탄소가 발생하기 때문에 차량의 이동이 잦은 곳에서는 최저 이산화탄소 농도가
높음
*/
// Lowest earth CO2 concentration 2020, in Hawaii is 413.4 ppm
// You can check this data at https://www.esrl.noaa.gov/gmd/ccgg/trends/
// but as you know, Hawaii don't emit mass CO2. so we add some number to Hawaii data.
// Where you are live in
//   - Big city and nearby huge traffic road: 40~80 ppm add to Hawaii ppm
//   - Big city and normal traffic road: 20~40 ppm
//   - Suburbs: 10~20 ppm
//   - 10 minutes on foot from traffic road, car can not reach: 0~10 ppm
//   normally gasoline engine makes huge carbon dioxide. if you use this sensor to indoor you should check
your environment like above.
const int32_t max_co2 = 6000;
/*  RX-9은 400~4000 ppm 의 농도에서 신뢰할 수 있는 수치를 나타냄
    그 이상의 농도에서도 측정은 가능하지만, 정확도가 떨어짐
    그래서 EXSEN은 4000 ppm 혹은 6000 ppm을 최대 농도로 제한할 것을 추천함
*/
// RX-9 is reliable 400 ~ 4000 ppm of CO2.
// RX-9 can measure 10000 ppm or more. but it shows little low accuracy.
// So, EXSEN recommend max_co2 as 6000 ppm.
//Timing Setting
int warm_up_time = 180; //default = 180, Rx-9 takes time to heat the core of sensor. 180 seconds is
require.
unsigned long current_time = 0;
unsigned long prev_time = 0;
int set_time = 1;  // default = 1, every 1 second, output the data.
```

```
/*
   특정 주기별로 센서의 값을 계산하고 출력하기 위한 Timing 관련 변수
   기본적으로 warm-up 시간은 3 분, 180 초로 함
   센서 출력 및 계산 주기는 1 초(set_time)으로 함
*/
//Moving Average
#define averaging_count 10 // default = 10, moving average count, if you want to see the data more
actively use lower number or stably use higher number
float m_averaging_data[averaging_count + 1][2] = {0,};
float sum_data[2] = {0,};
float EMF = 0;
float THER = 0;
int averaged_count = 0;
/*
   10 개의 데이터를 이동평균하는 것이 default
   센서의 변화가 더 빠르기를 원하는 경우 이 값(averaging_count)를 적은 수로 변경하고
   센서가 더 안정적인 값을 출력하기를 원하는 경우 이 값(averaging_count)를 더 큰 값을 변경
   이동평균은 E 핀과 T 핀에서 입력된 값에 대해서 수행함
   EMF: 이동평균 전 값, 현재 실시간 데이터
   THER: 이동평균 전 값, 현재 실시간 데이터
*/
//Sensor data
float EMF_data = 0;
float THER_data = 0;
float THER_ini = 0;
unsigned int mcu_resol = 1024;
float mcu_adc = 5;
/*
    센서 데이터를 저장하기 위한 변수
    EMF_data: 이동평균 후 값
    THER_data: 이동평균 후 값
    THER_ini: 써미스터의 기준 값, 3 분 워밍없이 끝난 직후의 값
*/
// Thermister constant
// RX-9 have thermistor inside of sensor package. this thermistor check the temperature of sensor to
compensate the data
// don't edit the number
#define C1 0.00230088
#define C2 0.000224
#define C3 0.00000002113323296
float Resist_0 = 15;
/*
   써미스터의 입력값을 섭씨로 변경하기 위한 상수값
   변경할 필요 없음
   Resist_0: 15kohm
*/
//Status of Sensor
bool status_sensor = 0;
/*
   센서 상태
   워밍업 이후에는 1, 이전에는 0
*/
//Step of co2
int CD1 = 700; // Very fresh, In Korea,
int CD2 = 1000; // normal
int CD3 = 2000; // high
int CD4 = 4000; // Very high
int status_step_CD = 0;
/*
   단계표시용 기준 농도
   400~700 ppm:   매우 좋음
```

```
   700~1000 ppm:  보통
   1000~2000 ppm: 높음
   2000~4000 ppm: 매우 높음
   수치는 사용하는 어플리케이션이나 상황에 따라 변경
   일반적으로 한국의 기준은
    기계환기인경우(환풍기를 사용하는 경우): 1000 ppm 이상인 경우 환기 권장
    자연환기인경우(창문을 열어서 환기하는 경우): 1500 ppm 이상인 경우 환기 권장
*/
// Standard of CO2 concenctration
/*
     South of Korea
        - Mechanical ventilation: 1000 ppm
        - natural ventilation: 1500 ppm
     Japan
        - School: 1500 ppm
        - Construction law: 1000 ppm
     WHO Europe: 920 ppm
     ASHRAE(US): 1000 ppm
     Singapore: 1000 ppm
*/
// CO2 to Human
/*
   < 450 ppm  : outdoor co2 concentration, Very fresh and healthy            NATURE
   ~ 700 ppm  : normal indoor concentration                                 HOME
   ~ 1000 ppm : no damage to human but sensitive person can feel discomfort,  OFFICE
   ~ 2000 ppm : little sleepy,                                              BUS
   ~ 3000 ppm : Stiff shoulder, headache,                                    METRO
   ~ 4000 ppm : Eye, throat irritation, headache, dizzy, blood pressure rise
   ~ 6000 ppm : Increased respiratory rate
   ~ 8000 ppm : Shortness of breath
   ~ 10000 ppm: black out in 2~3 minutes
   ~ 20000 ppm: very low oxygen exchange at lung. could be dead.
*/
//Calibration data
float cal_A = 372.1;        //Calibrated number
float cal_B = 63.27;        //Calibrated number
float cal_B_offset = 1.0;   //cal_B could be changed by their structure.
float co2_ppm = 0.0;
float co2_ppm_output = 0.0;
float DEDT = 1.0;           //Delta EMF/Delta THER, if DEDT = 1 means temperature change 1 degree in
Celsius, EMF value can changed 1 mV
/*
   cal_A: 교정 계수
   cal_B: 교정 계수
   cal_B_offset: 센서가 특정 기구물에 삽입되게 되면 기구물의 영향을 받게 되므로, 기구물 오프셋을 설정해줘야 함,
기구물이 완성된 다음에 농도테스트로 설정이 가능함
   co2_ppm: 계산된 co2 농도
   co2_ppm_output: 계산된 co2 농도를 min/max 를 내의 값으로 표현하기 위한 변수
   DEDT: delta EMF/delta THER, 온도에 따른 EMF 변화량
   RX-9 Simple 은 cal_A 와 cal_B 를 특별히 넣지 않고 특정값으로 계산함
   RX-9 은 cal_A 와 cal_B 값이 제품 뒤에 QR 코드로 명시가 되어 있으며, 해당 값을 제품마다 1:1 로 넣어서 ppm 을
계산할 수 있도록 해야 함
   알고리즘 내에서 cal_A 는 계속 변화하고 업데이트가 되는 반면 cal_B 는 변화하지 않음
*/
//Auto Calibration Coeff
unsigned long prev_time_METI = 0;
int MEIN = 120;                    //CAR: 120, HOME: 1440, Every MEIN minutes, Autocalibration is
executed.
int MEIN_common = 0;
int MEIN_start = 1;
bool MEIN_flag = 0;
```

```
int start_stablize = 300;
int METI = 60;                        //Every 60 second, check the autocalibration coef.
float EMF_max = 0;
float THER_max = 0;
int ELTI = 0;
int upper_cut = 0;
int under_cut_count = 0;
int under_cut_cnt_value = 300;
float under_cut = 0.99;               // if co2_ppm shows lower than (Base_line * undercut), sensor do re-calculation
/*
   prev_time_METI: 특정 주기적으로 자동보정을 실시하기 위한 변수
   MEIN: 센서를 자동차에서 사용하는 경우 120 분 주기로 실시, 일반가정인 경우 1440 분(하루, 24 시간) 주기로 실시
   MEIN_common: 설정한 MEIN 을 상황별로 변경해주기위한 변수, 예를 들어 센서가 데미지를 입었거나, 구동 초기인 경우 MEIN_common 의 값을 변경하여 자동보정 주기가 달라질 수 있도록 설정하여 사용함
   MEIN_start: 센서를 구동하기 시작한지 얼마 안 되었을 때, 센서값이 불안정한 경우가 있어서, 5 분 이내에는 자동보정이 1 분주기로 실행될 수 있도록함
   MEIN_flag: MEIN 의 값을 구동 초기 인지, 구동 초기가 아닌지에 따라서 변경해주기 위한 변수
   start_stablize: 센서를 장시간 비가동한 경우, 센서를 전원 인가 후 안정화 시퀀스를 돌리기 위한 시간변수, default = 300 초
   METI: 자동보정시 사용하는 변수, 자동보정을 하기 위해서 1 주기(MEIN 의 횟수) 내에 최대 EMF 값(최소 ppm 값, EMF 와 ppm 은 반비례 관계)과 최대 EMF 값일 때
         THER 값을 기억하여 MEIN 의 횟수가 만료될 때, 최대 EMF 값을 Base_line 에서 설정한 값으로 변경해주는데, 이 최대 EMF 값과 THER 값을 현재의 값과 비교하여
         더 높은 EMF 값을 기억하게 되는데, 이 비교하는 주기에 대한 수치임. 60 인 경우 60 초에 한번씩 EMF 값을 비교함.
         이 횟수는 MEIN 과 연동이 되기 때문에 하루 주기로 자동보정이 돌아가게 하기 위해서 METI 가 60 인경우 MEIN 이 1440 이라면, 1 주기는 1440 분
         이라, 하루마다 1 회씩 작동함.
         METI 가 30 인경우 하루 주기로 자동보정이 작동하게 하기 위해서는 MEIN 을 2880 으로 설정해야 함.
   EMF_max: 자동보정 1 주기 내에 METI 에서 설정한 수치마다 비교하여 기억하는 최대 EMF 값(최소 ppm 값)
   THER_max: EMF_max 를 업데이트할 때, 함께 기억하는 값. 실제로 THER 의 최대값은 아니고, EMF_max 값이 업데이트될 때, 함께 업데이트 됨
*/
// Damage recovery
// Sensor can be damaged from chemical gas like high concentrated VOC(Volatile Organic Compound), H2S, NH3, Acidic gas, etc highly reactive gas
// so if damage is come to sensor, sensor don't lose their internal calculation number about CO2.
// this code and variant are to prevent changing calculation number with LOCKING
bool damage_cnt_fg = 0;
unsigned int damage_cnt = 0;
unsigned int ppm_max_cnt = 0;
float cal_A_LOG[2][20] = {0,};
bool cal_A_LOCK = 0;
float cal_A_LOCK_value = 0.0;
float emf_LOCK_value = 0.0;
unsigned int LOCK_delta = 50;
unsigned int LOCK_disable = 5;
unsigned int LOCK_timer = 15;
unsigned int LOCK_timer_cnt = 0;
unsigned int S3_cnt = 0;
/*
   센서는 데미지를 받을 수 있습니다. 다양한 종류의 데미지가 존재할 수 있으나, 가장 일반적인 데미지는 VOC 와 같은 반응성이 높은 가스가
   센서로 고농도로 인입되는 경우 센서의 값이 흔들릴 수 있음
   이런 경우에 센서의 수치를 빠르게 회복시키기 위한 함수와 그에 대한 변수들.
   damage_cnt_fg: 센서가 데미지를 받아서 일반적인 실내에서 발생할 수 없는 높은 ppm 이 발생한 경우, 1 분 이상 해당 높은 농도가 유지된 경우
                  damage_cnt_fg 의 값을 1 로 변경함, 일반적인 경우 0
```

```
    damage_cnt: damage_cnt_fg 가 1 인 경우 자동보정 주기를 1 분마다 진행될 수 있도록 변경하고 자동 보정을 몇 회 진행했을 지를 카운트함
    ppm_max_cnt: damage_cnt_fg 에서 참조하고 있는 높은 농도가 유지된 경우 몇 초나 유지가 되어야 damage_cnt_fg 를 1 로 변경할지를 카운트함
    cal_A_LOG: 센서가 일시적으로 강한 데미지를 받았는지를 확인하기 위한 배열, 17 초 전 데이터와 현재 데이터를 비교하여 급격한 변화가 발생한 경우
               데미지를 받았다고 판단하기 위해 17 초 전 데이터를 보관함
*/
//debug
bool debug = 0;
bool display_mode = 0;
//command
const int timeout = 1000; //UART timeout
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  Serial.setTimeout(timeout);
  delay(1000);
  cal_A_LOCK = 0;
  damage_cnt_fg = 0;
  damage_cnt = 0;
}
void loop() {
  // put your main code here, to run repeatedly:
  /*
   *
   */
  current_time = millis() / 1000;
  if (current_time - prev_time >= set_time) {
    warm_up_chk();
    ppm_cal();
    DMG_REC();
    DMG_5000();
    step_cal_CD();
    auto_calib_co2();
    display_data();
    prev_time = current_time;
    /*
     *   Function       /               Description              /     Note           /
     *   warm_up_chk    / warming up timing chk                  / RX-9, RX-9 Simple   /
     *   ppm_cal        / calculation ppm value with EMF and THER   / RX-9, RX-9 Simple   /
     *   DMG_REC        / avoid damage from high reactivity gas     / RX-9, RX-9 Simple   /
     *   DMG_5000       / when sensor shows 5000 ppm, recovery      / RX-9, RX-9 Simple   /
     *   step_cal_CD    / step calculation, divided from ppm value  / RX-9 Simple         /
     *   auto_calib_co2 / periodically auto calibration             / RX-9, RX-9 Simple   /
     *   display_data   / data serial output, you can edit this     / Option             /
     *
     */
  }
}
void warm_up_chk() {
  if (current_time < warm_up_time) {
    status_sensor = 0;
  }
  else if (current_time >= warm_up_time && status_sensor == 0) {
    status_sensor = 1;
    sensor_reset();
  }
}
void ppm_cal() {
  /*
   * to calculate the concentration of CO2
```

```
   * 1. collet EMF(from E pin of RX-9) and THER(from T pin of RX-9)
   * 2. moving average to minimize noise
   * 3. calculate equation with EMF and THER
   * 4. limitate co2 value maximum and minimum
   * 5. if sensor shows too low co2 value, update the variant of calculation equation
   */
  EMF = analogRead(EMF_pin);
  EMF = EMF / mcu_resol; // 10 bits, Change the number if your MCU have other resolution
  EMF = EMF * mcu_adc;    // 5V     , Change the number if your MCU have other voltage
  EMF = EMF / 6;    //OPAMP   , Don't change!
  EMF = EMF * 1000; //V to mV conversion
  delay(1);
  THER = analogRead(THER_pin);
  THER = 1 / (C1 + C2 * log((Resist_0 * THER) / (mcu_resol - THER)) + C3 * pow(log((Resist_0 * THER) /
(mcu_resol - THER)), 3)) - 273.15;
  delay(1);
  // Moving Average START --->
  m_averaging_data[averaged_count][0] = EMF;
  m_averaging_data[averaged_count][1] = THER;
  if (averaged_count < averaging_count) {
    averaged_count++;
  }
  else if (averaged_count >= averaging_count) {
    for (int i = 0; i < averaging_count; i++) {
      sum_data[0] = sum_data[0] + m_averaging_data[i][0]; //EMF
      sum_data[1] = sum_data[1] + m_averaging_data[i][1]; //THER
      for (int j = 0; j < 2; j++) {
        m_averaging_data[i][j] = m_averaging_data[i + 1][j];
      }
    }
    EMF_data = sum_data[0] / averaging_count;
    THER_data = sum_data[1] / averaging_count;
    sum_data[0] = 0;
    sum_data[1] = 0;
  }
  // <---Moving Average END
  // CO2 Concentratio Calculation START --->
  co2_ppm = pow(10, ((cal_A - (EMF_data + DEDT * (THER_ini - THER_data))) / (cal_B * cal_B_offset)));
  co2_ppm = co2_ppm * 100 / 100;
  // <--- CO2 Concentration Calculation END
  // CO2 ppm min, max regulation START-->
  if (co2_ppm > max_co2) {
    co2_ppm_output = max_co2;
  }
  else if (co2_ppm <= Base_line) {
    co2_ppm_output = Base_line;
  }
  else {
    co2_ppm_output = co2_ppm;
  }
  // <--- CO2 ppm min, max regulation END
  // CO2 ppm baseline update START -->
  if (co2_ppm <= Base_line * under_cut) {
    under_cut_count++;
    if (under_cut_count > under_cut_cnt_value) {
      under_cut_count = 0;
      sensor_reset();
    }
  }
  else {
    under_cut_count = 0;
  }
  // <-- CO2 ppm baseline update END
```

```
}
void sensor_reset() {
  /*
   *  센서의 상태가 정상상태인 경우
   *  센서의 출력값을 초기화시킴
   *  센서의 정상상태 판단 (Warm-up 이 끝났을 것, DMG_REC()에서 판정하는 기준에 해당하지 않을 것)
   *
   *
   */
  if (cal_A_LOCK == 0) {
    THER_ini = THER_data;
    EMF_max = EMF_data;
    THER_max = THER_data;
    ELTI = 0;
    cal_A = EMF_data + log10(Base_line) * (cal_B * cal_B_offset);
  }
}
void step_cal_CD() {
  /*
   *  RX-9 Simple 에 해당
   *  CD1~CD4 에서 지정한 값에 해당할 경우 센서의 단계를 표현함
   *  본 함수에서는 0~4 의 총 5 단계로 구분됨
   *  ppm 출력을 하지 않는 RX-9 Simple 의 경우에도 ppm_cal()이 필요한 이유가
   *  다음의 단계 출력을 위해 ppm 값을 사용하기 때문임
   *
   */

  if (status_sensor == 1) {
    if (co2_ppm < CD1) {
      status_step_CD = 0;
    }
    else if (co2_ppm >= CD1 && co2_ppm < CD2) {
      status_step_CD = 1;
    }
    else if (co2_ppm >= CD2 && co2_ppm < CD3) {
      status_step_CD = 2;
    }
    else if (co2_ppm >= CD3 && co2_ppm < CD4) {
      status_step_CD = 3;
    }
    else if (co2_ppm >= CD4) {
      status_step_CD = 4;
    }
  }
}
void auto_calib_co2() {
  /*
   * 자동보정함수
   * 모든 센서가 센서를 구동하다 보면 센서의 출력값이 정상 범위를 벗어나기 마련인데,
   * 이산화탄소 센서는 영점 조정을 해줄만한 좋은 레퍼런스가 있기 때문에 자동보정으로 0 점 조정을 할 수 있음
   * 좋은 레퍼런스는 지구의 이산화탄소 농도인데, 일일간 최저농도의 편차가 크지 않기 때문에 일정 기간 중 측정된 값의
최저값을
   * 지그의 이산화탄소 농도에 맞춰 0 점 조정을 진행함.
   * 모든 이산화탄소 센서는 이와 같은 자동보정을 하고 있음.
   *
   * ELTI(Elapsed Time): 자동보정을 위한 시간을 카운트함
   * MEIN(Measurement Interval): 자동보정의 1 주기 횟수
   * METI(Measurement Time): 측정 주기
   *
   * 주요 동작 시퀀스
```

```
   * 1. METI 에서 설정한 측정주기마다 EMF_max 값과 현재의 EMF 값을 비교함
   * 2-1. EMF_max 보다 현재의 EMF 값이 크면 3 회 동안 추이를 보고 3 회 연속 큰 경우, EMF_max 값을 현재의 EMF 값으로
업데이트함, 이 때 THER_max 값을 현재의 THER 값으로 함께 업데이트함
   * 2-2. EMF_max 보다 현재의 EMF 값이 작으면 EMF_max 값을 업데이트하지 않고 그대로 둠
   * 3. 2 에서 EMF_max 와 EMF 현재값을 비교할 때마다 ELTI 는 1 씩 증가함
   * 4. ELTI 가 MEIN 에서 설정한 횟수가 되면 자동보정을 실시함
   * 5. 자동보정은 MEIN 에서 설정한 1 주기(횟수) 동안 비교한 EMF 값 중 가장 높은 값(EMF_max)을 지구대기
농도(Base_line)로 환산함
   * 6. 다음주기를 시작함, 관련변수를 현재값으로 초기화함
   *
   * 기타
   * cal_A_LOCK: 센서가 데미지를 받은 상황이라면, autocal 이 진행되지 않음
   *
   */
  if (current_time < start_stablize && MEIN_flag == 0) {
    MEIN_flag = 1;
    MEIN_common = MEIN_start;
  }
  /*
   *
   */
  else if (current_time >= start_stablize + 1 && MEIN_flag == 1 && damage_cnt_fg == 0) {
    MEIN_common = MEIN;
    //if(display_mode){Serial.println("MEIN_common = MEIN");}
  }
  if (current_time - prev_time_METI >= METI && status_sensor == 1) {
    if (ELTI < MEIN_common) {
      if (cal_A_LOCK == 0) {
        ELTI++;
      }
      else {
        LOCK_timer_cnt++;
      }
    }
    else if (ELTI >= MEIN_common) {
      if (cal_A_LOCK == 0) {
        cal_A = (EMF_max + DEDT * (THER_max - THER_data)) + log10(Base_line) * (cal_B * cal_B_offset);
        THER_ini = THER_data;
        EMF_max = EMF_data;
        THER_max = THER_data;
        ELTI = 0;
      }
      if (damage_cnt_fg == 1)
      {
        damage_cnt++;
      }
    }
    if (EMF_max >= EMF_data) {
      upper_cut = 0;
    }
    else if (EMF_max < EMF_data) {
      upper_cut++;
      if (upper_cut > 3) {
        EMF_max = EMF_data;
        THER_max = THER_data;
        upper_cut = 0;
      }
    }
    prev_time_METI = current_time;
  }
}
```

```
void DMG_REC() {
  /*
   * 데미지 감지 함수
   * 센서가 데미지를 받으면 특히, 반응성이 좋은 가스(에탄올, 포름알데히드, 암모니아, CO 등)에 노출되었을 때, 센서의 EMF 값이 급격하게 증가할 수 있음
   * 이런 경우에 데미지를 받았는지 판정하고 데미지를 받았다고 판단된 경우, 센서를 보정한 계수(cal_A)의 업데이트를 멈추고, 데미지를 받기 전의 cal_A 를 적용하여
   * 데미지에 의해서 센서의 보정계수가 바뀌지 않도록 하는 함수
   *
   * 주요시퀀스
   * 1. 최근 20 초간 EMF, cal_A 값을 저장함
   * 2. 센서가 워밍업을 지난 시간이라면 현재의 EMF 와 17 초 전 EMF 를 비교함
   * 3-1. 3 초 연속으로 17 초전 EMF 와 현재의 EMF 차이가 LOCK_delta 에서 지정한 값보다 크면 cal_A 를 업데이트하는 것을 정지하고 19 초전의 cal_A 값과, EMF 값을 기억함
   * 3-2. EMF 가 최대값에 근접한 경우 3-1 이 작동하지 않기 때문에 최대값 한정 조건문으로 최대값에 가깝더라도 급격하게 바뀐 경우 작동하도록 함
   * 4. 3-1 이나 3-2 에서 cal_A_LOCK = 1 이 된 경우 다음의 두 조건으로 cal_A_LOCK 을 해제함
   * 4-1. 3-1 에서 기억한 EMF 값과 현재의 EMF 값의 차이가 LOCK_disable 에서 지정한 값보다 작아지는 경우 cal_A_LOCK 을 해제함
   * 4-2. cal_A_LOCK 이 설정된 이후 시간이 LOCK_timer(단위: 분)에서 설정한 값보다 커지면 cal_A_LOCK 을 해제함
   */
  for (int i = 0; i < 19; i++) {
    cal_A_LOG[0][i] = cal_A_LOG[0][i + 1];
    cal_A_LOG[1][i] = cal_A_LOG[1][i + 1];
  }
  cal_A_LOG[0][19] = cal_A;
  cal_A_LOG[1][19] = EMF_data;
  if (status_sensor == 1) {
    if ((cal_A_LOG[1][19] - cal_A_LOG[1][2] >= LOCK_delta) && (cal_A_LOG[1][18] - cal_A_LOG[1][1] >= LOCK_delta) && (cal_A_LOG[1][17] - cal_A_LOG[1][0] >= LOCK_delta)) {
      if (cal_A_LOCK == 0) {
        cal_A_LOCK = 1;
        cal_A_LOCK_value = cal_A_LOG[0][0];
        emf_LOCK_value = cal_A_LOG[1][0];
        cal_A = cal_A_LOCK_value;
        //if(debug); Serial.println("S1 ---- cal_A_LOG[1][0] = " + cal_A_LOG[1][0] + "cal_A_LOG[1][9] = " + cal_A_LOG[1][9]);
      }
    }
    else if ((cal_A_LOG[1][2] > 540 - LOCK_delta) && (cal_A_LOG[1][1] > 540 - LOCK_delta) && (cal_A_LOG[1][0] > 540 - LOCK_delta) && (cal_A_LOG[1][2] <= 540 - LOCK_disable) && (cal_A_LOG[1][1] <= 540 - LOCK_disable) && (cal_A_LOG[1][0] <= 540 - LOCK_disable)) {
      if ((cal_A_LOG[1][17] > 540) && (cal_A_LOG[1][18] > 540) && (cal_A_LOG[1][19] > 540)) {
        if (cal_A_LOCK == 0) {
          cal_A_LOCK = 1;
          cal_A_LOCK_value = cal_A_LOG[0][0];
          emf_LOCK_value = cal_A_LOG[1][0];
          cal_A = cal_A_LOCK_value;
          //if(debug); Serial.println("S2 ---- cal_A_LOG[1][0] = " + cal_A_LOG[1][0] + "cal_A_LOG[1][9] = " + cal_A_LOG[1][9]);
        }
      }
    }
    else {
      //do nothing
    }
  }
  if (cal_A_LOCK == 1) {
    if (EMF_data - emf_LOCK_value < LOCK_disable) {
      S3_cnt++;
      if (S3_cnt >= 10) {
```

```
        S3_cnt = 0;
        cal_A_LOCK = 0;
        ELTI = 0;
        THER_ini = THER_data;
        EMF_max = EMF_data;
        THER_max = THER_data;
        LOCK_timer_cnt = 0;
      }
      else {
        //do nothing
      }
    }
    else if (LOCK_timer_cnt >= LOCK_timer) {
      cal_A_LOCK = 0;
      ELTI = 0;
      THER_ini = THER_data;
      EMF_max = EMF_data;
      THER_max = THER_data;
      LOCK_timer_cnt = 0;
    }
    else {
      S3_cnt = 0;
    }
  }
  else {
    //do nothing
  }
}
void display_data() {
  /*
   * 데이터 표시
   */
  if (display_mode == 0) {
    Serial.print("# ");
    if (co2_ppm <= 999) {
      Serial.print("0");
      Serial.print(co2_ppm_output, 0);
    }
    else {
      Serial.print(co2_ppm_output, 0);
    }
    Serial.print(" ");
    if (status_sensor == 0) {
      Serial.print("WU");
    }
    else {
      Serial.print("NR");
    }
    Serial.println("");
  }
  else if (display_mode == 1) {
    Serial.print("T ");
    Serial.print(current_time);
    Serial.print(" # ");
    if (co2_ppm <= 999) {
      Serial.print("0");
      Serial.print(co2_ppm, 0);
    }
    else {
      Serial.print(co2_ppm, 0);
    }
    Serial.print(" | CS: ");
    Serial.print(status_step_CD);
```

```
    Serial.print(" | ");
    Serial.print(EMF_data);
    Serial.print(" mV | ");
    Serial.print(THER_data);
    if (status_sensor == 0) {
      Serial.print(" oC | WU");
    }
    else {
      Serial.print(" | NR");
    }
    Serial.println("");
  }
}
void DMG_5000()
{
  /* 데미지 회복 함수
   * 일반적으로 발생하기 어려운 5000 ppm 이상의 CO2 가 1 분 이상 유지된 경우, 센서가 데미지를 받은 것이라 판단하여
   * 자동보정주기를 3 분(2 로 표기)으로 변경하여 센서가 데미지로부터 빠르게 회복할 수 있도록 하고, 3 분 간격으로
자동보정을 5 회 진행한 후에는
   * 다시 원래의 주기로 복귀함
   *
   * 1. 현재 출력되는 co2 농도가 5000 ppm 이상인 경우 1 분을 체크함
   * 2. 1 분 이상 5000 ppm 이상이 유지된 경우 damage_cnt_fg 를 1 로 변경하여 damage_cnt 를 증가하게 두고
damage_cnt 가 5 를 초과할 때까지 자동보정을 3 분 주기로 진행함, 총 15 분(damage_cnt * 3 분) 소요
   */
  if (status_sensor == 1) {
    if (co2_ppm_output >= 5000) {
      if (ppm_max_cnt > 60) {
        MEIN_common = 2;
        damage_cnt_fg = 1;
        ppm_max_cnt = 0;
      }
      else {
        ppm_max_cnt++;
      }
    }
    else {
      ppm_max_cnt = 0;
    }
  }
  if (damage_cnt > 5) {
    MEIN_common = MEIN;
    damage_cnt = 0;
    damage_cnt_fg = 0;
  }
  else {
    //do nothing
  }
}
```