로봇학실험 3 Smart Planter
결과보고서

2018741035 한준호
2018741038 황유환

1. 서론

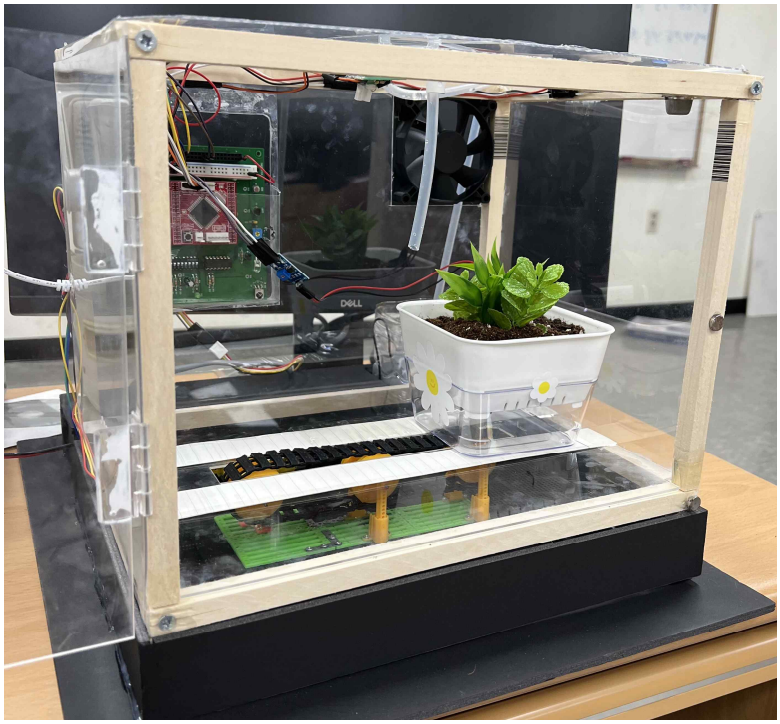
코로나가 점점 심해짐에 따라 집 밖을 나가기 꺼려하는 사람들이 많아졌고 그에따라 사람들이 스트레스를 해소하기 위해 많이 했던 행동중 하나가 바로 집에서 식물을 키우는 것입니다.
비록 코로나 때문만이 아니라 점점 식물을 키우는 사람들이 많아짐에 따라 반려 식물이라는 말도 생겨나고 있는데 바쁜 현대인들에게 특수한 경우가 아니라면 대다수의 사람들은 그 식물만을 보고 있기 힘든 상황일 것이고 매일 한번씩 식물을 꼼꼼히 확인하기 힘들고 어쩔 수 없이 식물을 방치할 수도 있습니다.
키우면서 애정이 들어 반려 식물이라는 단어도 생기는 상황 속에서 식물들의 특성상 교감이 쉽지 않기 때문에 잘 키울 수 있게 도와주기 위해 매일 꼼꼼히 확인하지 않더라도 한눈에 식물에게 필요한 것과 디테일한 부분을 케어해주는 'Smart Planter'를 이번 작품으로 선정하게 되었습니다.

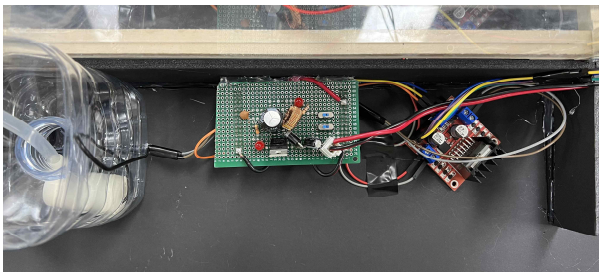
구상한 기능은 크게 다음과 같습니다.

1. 어두워지면 안에 있는 식물이 잘 보이게 led ON
2. 담배 연기 등 유해가스가 들어왔을 때 팬을 돌려 내부 환기
3. 토양에 수분이 부족할 때 수분공급
4. 화분의 크기에 맞게 화분의 위치와 수분 공급원 위치 맞추기
5. led를 통해 내부 온도를 한눈에 파악
6. 화분 위치 조절에 쓰이는 컨베이어 벨트 속도 조절

<시나리오>
먼저 문을 열어 화분을 적당한 위치에 넣는다. 전원을 켜면 psd 센서가 거리를 측정해 화분의 위치가 수분 공급원과 맞게끔 컨베이어 벨트가 움직여 위치를 맞춘다. 속도가 너무 빠르거나 느리면 가변저항을 조절해 컨베이어 벨트의 속도를 조절한다. 만약 흙이 마르면 위에서 물이 나오고, 유해가스가 들어오면 팬이 작동하여 내부를 환기한다. 온도에 따라 메인보드의 led가 올라가게 만들어 실내의 온도를 한눈에 파악할 수 있고, 만약 어두워지면 천장의 led가 켜지게 되어 야간에 식물이 보이지 않는 상황을 방지할 수 있고, 야간 감수성 느낌으로 색다른 느낌의 식물을 즐길 수 있다.

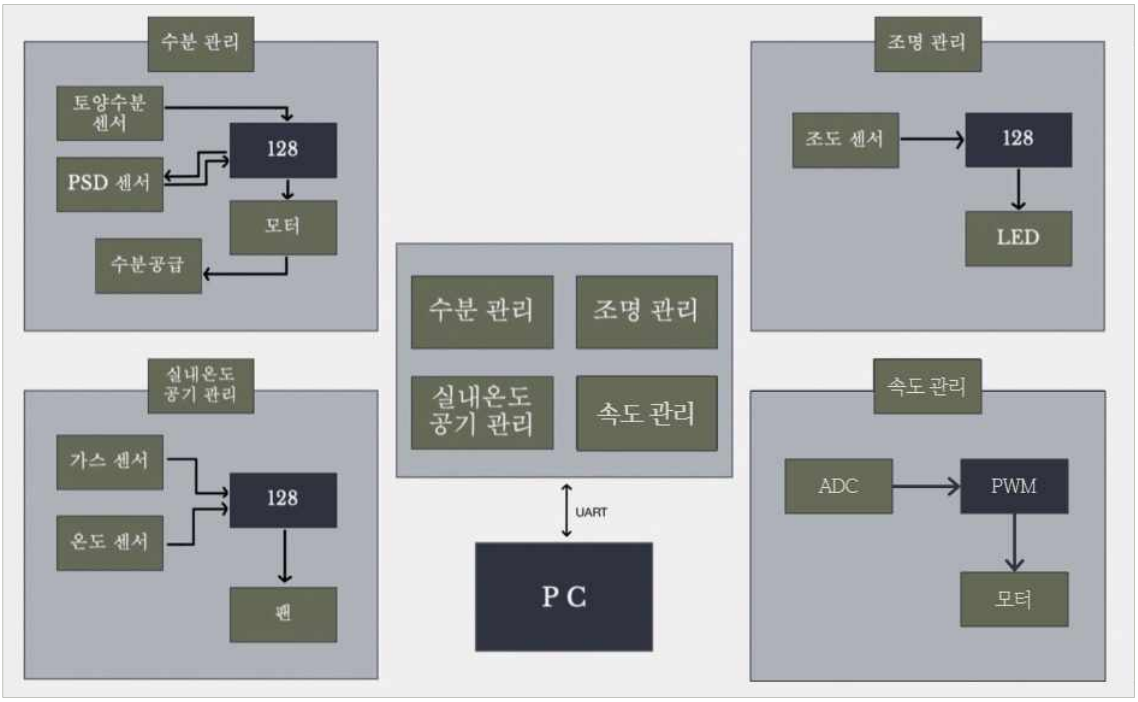
2. 하드웨어 구성





하드웨어 구성은 위의 사진과 같다.
천장에는 가스를 탐지하는 가스 센서와 물을 주는 워터 파이프라인 그리고 조명 역할을 해주는 led가 있고 정면에는 열고 닫을 수 있는 문이 있다. 후방에는 환기를 시켜주는 팬과 메인보드가 위치하는 두꺼비집이 있으며 측면에는 화분의 거리를 측정할 psd센서가 부착되어있다. 바닥은 크게 2층으로 구성되어 있는데 내부 1층에는 화분의 위치를 조정해줄 컨베이어 벨트와 외부의 바닥에는 전원부와 모터드라이브, 물탱크가 있으며 내부 2층에는 화분이 있다.

3. 시스템 설명



<System Architecture>

▶ Moving Average Filter & Low Pass Filter

```
double Low_Pass_Filter(double data)
{
    LPF = (dt*data+LPF_tau*LPF_past)/(LPF_tau+dt);
    LPF_past = LPF;

    return LPF;
}
```

```
double Moving_Average_Filter(double data)
{
    MAF = 0;
    MAF_sample[m] = data;
    int j=0;
    for(j=0;j<10;j++)
    {
        MAF += MAF_sample[j];
    }
    if (m==9) m=0;
    else m++;

    return (MAF/10);
}
```

Moving average filter와 Low pass filter는 수업 시간에 진행한 방식대로 위의 코드와 같이 만들어 주었다

```
double LPF_adc = Low_Pass_Filter(adc);
double MAF_adc = Moving_Average_Filter(adc);
double LPF_cds = Low_Pass_Filter(cds);
double MAF_cds = Moving_Average_Filter(cds);
double LPF_lm35 = Low_Pass_Filter(lm35);
double MAF_lm35 = Moving_Average_Filter(lm35);
double LPF_psd = Low_Pass_Filter(psd);
double MAF_psd = Moving_Average_Filter(psd);
double LPF_moist = Low_Pass_Filter(moist);
double MAF_moist = Moving_Average_Filter(moist);
double LPF_gas = Low_Pass_Filter(gas);
double MAF_gas = Moving_Average_Filter(gas);
```

```
UART_TX_string(DEC_TO_CHAR(adc));   UART_TX(',');
UART_TX_string(DEC_TO_CHAR(cds));   UART_TX(',');
UART_TX_string(DEC_TO_CHAR(lm35));  UART_TX(',');
UART_TX_string(DEC_TO_CHAR(psd));   UART_TX(',');
UART_TX_string(DEC_TO_CHAR(moist)); UART_TX(',');
UART_TX_string(DEC_TO_CHAR(gas));  UART_TX(',');
UART_TX_string(DEC_TO_CHAR(fast));  UART_TX(',');

UART_TX_string(DEC_TO_CHAR(LPF_gas));   UART_TX(',');
UART_TX_string(DEC_TO_CHAR(MAF_gas));   UART_TX(',');
```

위의 코드와 serial chart를 활용해 원래 값과 filtering 결과값들을 비교 해 보았다.

▶ 가변저항

```
//***ADC
double get_ADC(void)
{
    ADMUX = 0x00;
    ADCSRA |= (1<<ADSC);//0x40;
    while(!(ADCSRA & (1<<ADIF)));
    int adc = ADC;
    double Vadc = adc*5./1023.;

    return Vadc;
}
```

0~1023까지의 ADC값을 0~5V 전압으로 바꿔주었다.

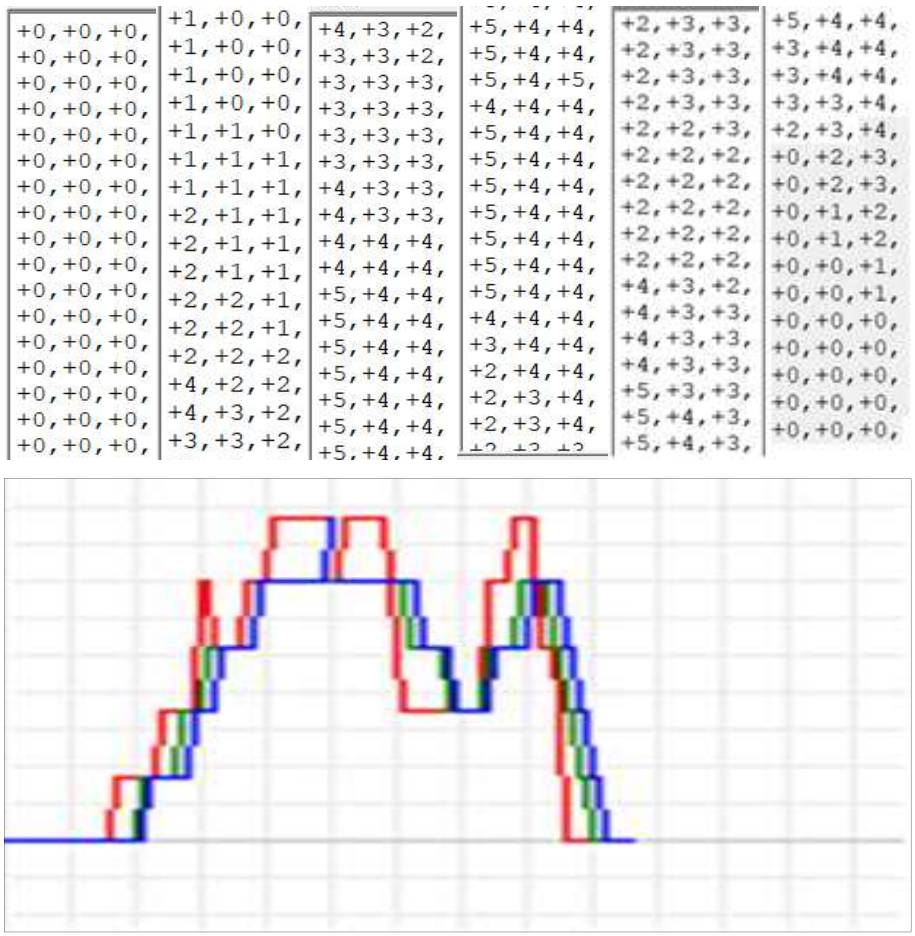
```
//ADC
void adc_func(double num)
{
    ICR1 = 312;
    if(num < 1)
    {
        fast = 0;
    }
    else if (num < 2 )
    {
        fast = 128;
    }
    else if (num < 3 )
    {
        fast = 130;
    }
    else if (num < 4 )
    {
        fast = 132;
    }
```

```
    else if (num < 5 )
    {
        fast = 200;
    }
    else if (num < 6 )
    {
        fast = 280;
    }
    else
    {
        fast = 0;
    }
}
```

우리가 사용하는 화분이 아닌 다른 화분을 쓸 경우 무게가 달라진다. 이에 가변저항을 이용해 컨베이어 벨트의 속도를 조절할 수 있게 하여 무게가 다른 화분에도 위치조정이 적용될 수 있도록 하였다.
가변저항의 값에 따라 컨베이어 벨트의 속도를 조절하기 위해 위와 같이 코드를 작성하였다. 가변저항은 0~5 사이의 정수값이 나오고 Compare match mode를 사용하기 때문에 Top값 = ICR1 = 312. fast를 중간값으로 설정하고 각각의 값에따라 fast 값을 조절해 속도를 제어하였다.

<Filtering>

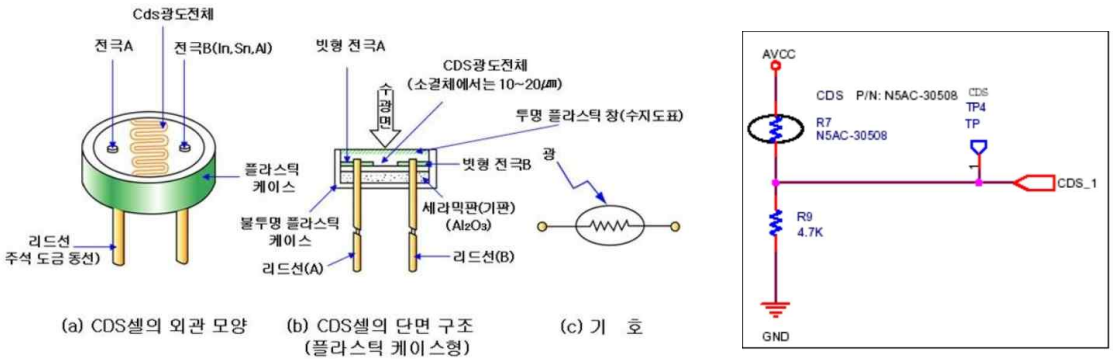
위에서 설명한 Low pass filter와 Moving average filter를 사용해 serial chart로 실제값과 필터값을 받아본 결과 다음과 같았다.



< ● 기본값 ● Low pass filter ● moving average filter>

그래프를 확인해본 결과 Low pass filter를 사용하기로 하였다.

▶ 조도센서(GL5537) CDS



반도체에 빛이 닿으면 전자와 정공이 증가하고 조사된 빛 에너지에 비례하여 전류가 증가하는 원리를 이용하였다. 조도가 높아질수록 저항이 작아지는 가변저항이라고 생각하면 된다.

우리가 사용하는 모델은 GL5537으로 데이터 시트를 보면 Vmax = 150, Pmax = 100, Ambient Temp = $-30^\circ \sim +70^\circ$, Spectral Peak = 540(nm), Photo Resistance는 R10 = 20~50($k\Omega$)이고 R100 = 4~10($k\Omega$)이다. Dark Resistance(closed 10 lux 10초뒤 측정) = 2.0$M\Omega$이며 Response Time은 Rise Time = 20(ms), Descent Time = 30(ms)이다.

관계식을 보면 $R_{cds} = \dfrac{R_9 \times AVCC}{V_{out}} - R_9$의식을 얻을 수 있다. 옆의 회로도를 보면 R_9=4.7K이고 AVCC는 5V, V_{out} = 우리가 CDS 센서에서 받는 값(num)이 된다.

$$\gamma = \frac{\log(R_{cds}[\Omega]) - \log(35[k\Omega])}{\log(10[Lux]) - \log(x[Lux])} \rightarrow \log(x[Lux]) = 1 - \frac{\log(R_{cds}[\Omega]) - \log(35[k\Omega])}{\gamma}$$

<데이터 시트>

model	V max (VDC)	P max (mW)	Ambient Temp(°C)	Spectral Peak(nm)	Photo Resistance (kΩ)		Dark Resistance (MΩ)	100 10	Response Time(ms)		Illuminance Vs photo resistance
					R10	R100			Rise tirne	descent tirne	
GL5537	150	100	-30~+70	540	20—50	4—10	2.0	0.7	20	30	4

이므로 γ = 0.7이고 위에서 얻은 R_{cds} 값을 사용하면 조도값을 구할 수 있다.

```
double get_CDS(void)
{
    ADMUX = 0x01;
    ADCSRA |= (1<<ADSC);
    while(!(ADCSRA & (1<<ADIF)));
    int adc = ADC; //adc: 0~1023
    double Vadc = adc*5./1023.; // Vadc: 0~5

    return Vadc; //[V]
}

double cds_calculate(double num)
{
    double Rcds = 5*4700/num - 4700;
    long y = 1- (log(Rcds)-log(35000.))/0.7;
    double x = 10^y;
    return x; //[LUX]
}
```

```
//CDS
void cds_func(double num)
{
   if(num > 12)
   {
       led_on = 0; //천장 조명 off
   }
   else
   {
       led_on = 1; //천장 조명 on
   }
}
```

flag check를 위해 while(!(ADCSRA & (1<<ADIF)));를 해주었고 0~1023까지의 수치로 나오는 ADC값을 0~5[V]값으로 변환시키기 위해 Vadc = adc*5./1023. 으로 식을 구성했다. (double 이므로 5. 와 1023.으로 표현)

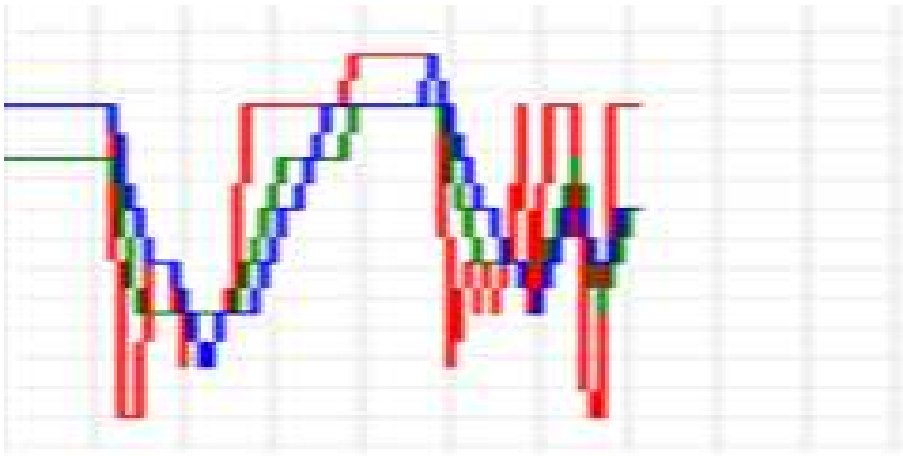
위에서 설명한 조도를 구하는 식을 사용해 0~5[V]로 표현된 ADC값을 조도값[LUX]으로 변환

led도 PORTC에 연결되어 있는데 시스템 팬과 수중 모터 또한 PORTC를 사용하는데 여러 조건에 따라 각각의 소자들을 상충 되지 않게 작동시키기 위해 led_on이라는 flag 변수를 활용하였다.
<자세한 제어 방법은 뒤에 서술>

<Filtering>

위에서 설명한 Low pass filter와 Moving average filter를 사용해 serial chart로 실제값과 필터값을 받아본 결과 다음과 같았다.

+14,+13,+14,	+11,+10,+11,	+14,+12,+11,	+15,+14,+14,	+11,+11,+11,	+14,+13,+12,
+14,+13,+14,	+11,+10,+11,	+14,+13,+12,	+15,+14,+14,	+14,+11,+11,	+9,+12,+12,
+14,+13,+14,	+11,+10,+11,	+14,+13,+12,	+15,+14,+15,	+10,+11,+10,	+8,+11,+11,
+14,+13,+14,	+11,+10,+11,	+14,+13,+12,	+14,+14,+14,	+11,+11,+10,	+8,+10,+11,
+14,+13,+14,	+9,+10,+10,	+14,+13,+13,	+9,+13,+14,	+14,+11,+11,	+14,+11,+11,
+14,+13,+14,	+10,+10,+10,	+14,+13,+13,	+10,+12,+13,	+14,+12,+11,	+14,+11,+12,
+14,+13,+14,	+10,+10,+9,	+14,+13,+14,	+11,+12,+13,	+14,+12,+12,	+14,+11,+12,
+14,+13,+14,	+10,+10,+9,	+14,+13,+14,	+10,+11,+12,	+14,+13,+12,	+14,+12,+12,
+14,+13,+14,	+10,+10,+10,	+14,+13,+14,	+11,+11,+12,	+9,+12,+12,	+14,+12,+12,
+14,+13,+14,	+10,+10,+10,	+15,+14,+14,	+10,+11,+12,	+8,+11,+11,	+14,+12,+12,
+14,+13,+14,	+11,+10,+10,	+15,+14,+14,	+11,+11,+11,	+8,+10,+11,	+14,+12,+12,
+14,+13,+14,	+14,+11,+10,	+15,+14,+14,	+11,+11,+11,	+14,+11,+11,	+14,+12,+12,
+14,+13,+14,	+14,+11,+10,	+15,+14,+14,	+14,+11,+11,	+14,+11,+12,	+14,+12,+12,
+8,+12,+13,	+14,+12,+11,	+15,+14,+14,	+10,+11,+10,	+14,+12,+12,	+14,+12,+12,
+8,+11,+12,					
+8,+10,+12,					



< ● 기본값 ● Low pass filter ● moving average filter>

그래프를 확인해본 결과 Low pass filter를 사용하기로 하였다.

▶ LM35

'Smart Planter' 내부에 온도를 LED를 통해 확인하기 위해 내부 온도를 측정하는 센서이다

LM35 Datasheet

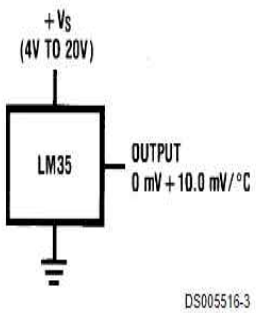
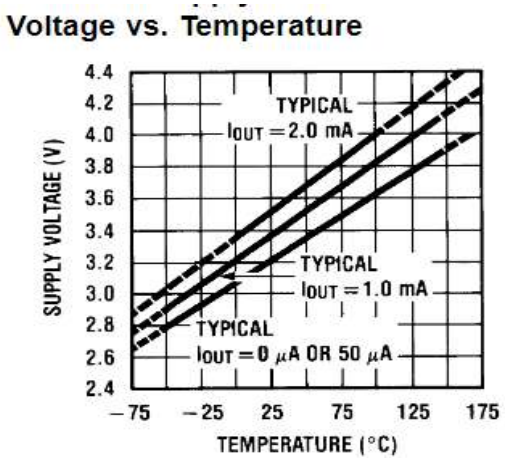


FIGURE 1. Basic Centigrade Temperature Sensor
(+2°C to +150°C)




```
//***LM35 온도센서
double get_LM35(void)
{
    ADMUX = 0x02;
    ADCSRA |= (1<<ADSC);//0x40;
    while(!(ADCSRA & (1<<ADIF)));
    int adc = ADC;
    double Vadc = adc*5./1023.;

    return Vadc;
}
```

- 온도 센서를 통해 입력받아 변하는 0~1023 까지의 bit값을 0~5V로 환산하여 출력

```
double LM35_calculate(double num)
{
    double T = 100*num;
    return T; //['c]
}
```

-데이터시트를 보면 전압과 온도 사이에 비례적인 관계이므로 0~5V값을 C° 값으로 환산

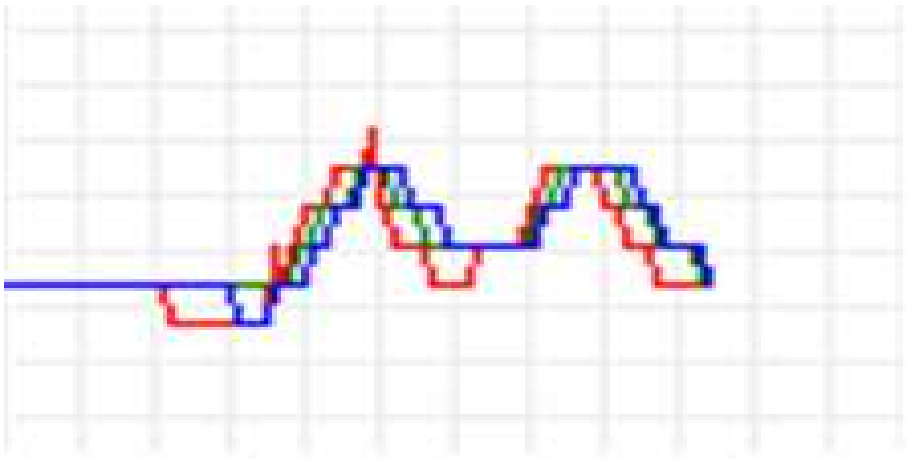
```
//LM35
void lm35_func(double num)
{
    if(num <= 18)
    {
        PORTA = 0b11111110;
    }
    else if(num <= 20)
    {
        PORTA = 0b11111100;
    }
    else if(num <= 22)
    {
        PORTA =0b11111000;
    }
    else if(num <= 24)
    {
        PORTA = 0b11110000;
    }
    else if(num <= 26)
    {
        PORTA = 0b11100000;
    }
    else if(num <= 28)
    {
        PORTA = 0b11100000;
    }
    else if(num <= 30)
    {
        PORTA = 0b11000000;
    }
    else
    {
        PORTA = 0b00000000;
    }
}
```

- 온도에 따라 LED가 다르게 작동

<Filtering>

위에서 설명한 Low pass filter와 Moving average filter를 사용해 serial chart로 실제값과 필터값을 받아본 결과 다음과 같았다.

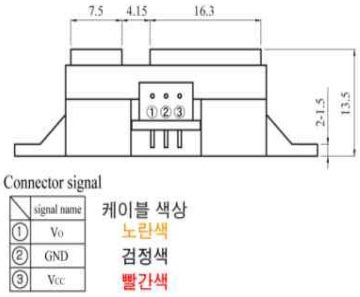
+22,+22,+22,	+22,+22,+22,	+21,+22,+22,	+25,+24,+24,	+22,+23,+24,	+25,+24,+24,
+22,+22,+22,	+22,+22,+22,	+21,+22,+22,	+25,+24,+24,	+22,+23,+23,	+25,+24,+24,
+22,+22,+22,	+22,+22,+22,	+21,+22,+21,	+25,+24,+24,	+22,+23,+23,	+25,+25,+24,
+22,+22,+22,	+22,+22,+22,	+21,+22,+21,	+25,+25,+24,	+22,+23,+23,	+25,+25,+24,
+22,+22,+22,	+22,+22,+22,	+21,+22,+21,	+25,+25,+25,	+22,+23,+23,	+25,+25,+25,
+22,+22,+22,	+22,+22,+22,	+21,+22,+21,	+26,+25,+25,	+23,+23,+23,	+25,+25,+25,
+22,+22,+22,	+22,+22,+22,	+21,+22,+21,	+24,+25,+25,	+23,+23,+23,	+25,+25,+25,
+22,+22,+22,	+21,+22,+22,	+23,+22,+22,	+24,+25,+25,	+23,+23,+23,	+25,+25,+25,
+22,+22,+22,	+21,+22,+22,	+22,+22,+22,	+23,+24,+25,	+23,+23,+23,	+24,+25,+25,
+22,+22,+22,	+21,+22,+22,	+23,+22,+22,	+23,+24,+25,	+23,+23,+23,	+24,+25,+25,
+22,+22,+22,	+21,+22,+22,	+23,+23,+22,	+23,+24,+24,	+23,+23,+23,	+24,+25,+25,
+22,+22,+22,	+21,+22,+22,	+24,+23,+22,	+23,+24,+24,	+23,+23,+23,	+23,+24,+25,
+22,+22,+22,	+21,+22,+22,	+24,+23,+23,	+23,+23,+24,	+24,+23,+23,	+23,+24,+25,
+22,+22,+22,	+21,+22,+22,	+24,+24,+23,	+22,+23,+24,	+24,+24,+23,	+23,+24,+24,
+22,+22,+22,	+21,+22,+22,	+24,+24,+23,		+25,+24,+24,	+23,+24,+24,



< ● 기본값 ● Low pass filter ● moving average filter>

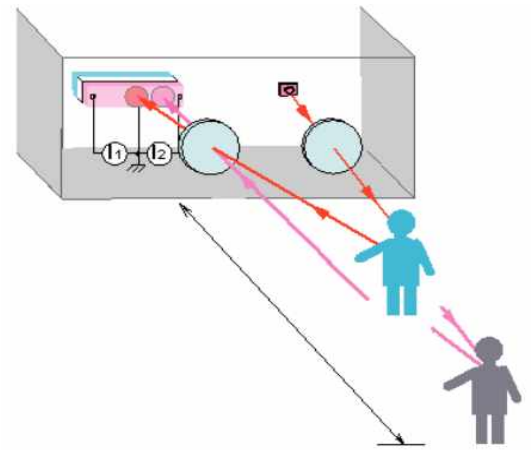
그래프를 확인해본 결과 Low pass filter를 사용하기로 하였다.

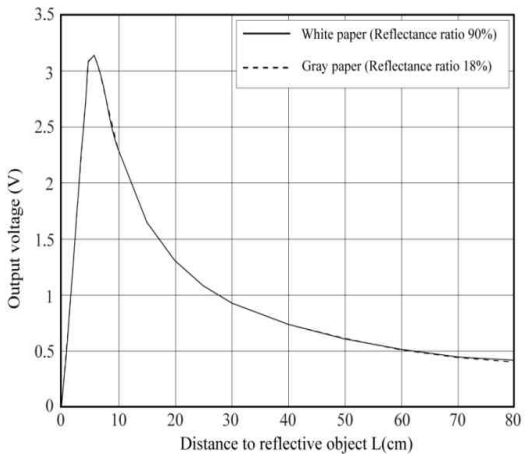
▶ PSD(C29)



화분의 크기에 맞게 화분의 위치와 수분 공급원 위치 맞추기 위해 화분과 PSD센서 사이의 거리를 측정하는 센서이다.

C29 Datasheet





10~80cm 범위의 거리를 측정할 수 있는 것을 알 수 있다.

```
/***PSD 센서
double get_PSD(void)

  ADMUX = 0x04;
  ADCSRA |= (1<<ADSC);//0x40;
  while(!(ADCSRA & (1<<ADIF)));
  int adc = ADC;
  double Vadc = adc*5./1023.;

  return Vadc;
```

- PSD 센서를 통해 입력받아 변하는 0~1023 까지의 bit값을 0~5V로 환산하여 출력

```
double PSD_calculate(double num)
{
   double distance = (27.61/ (num - 0.1696));
   if(distance<10)
   {
    distance = 10;
   }
   return distance; //[cm]
}
```

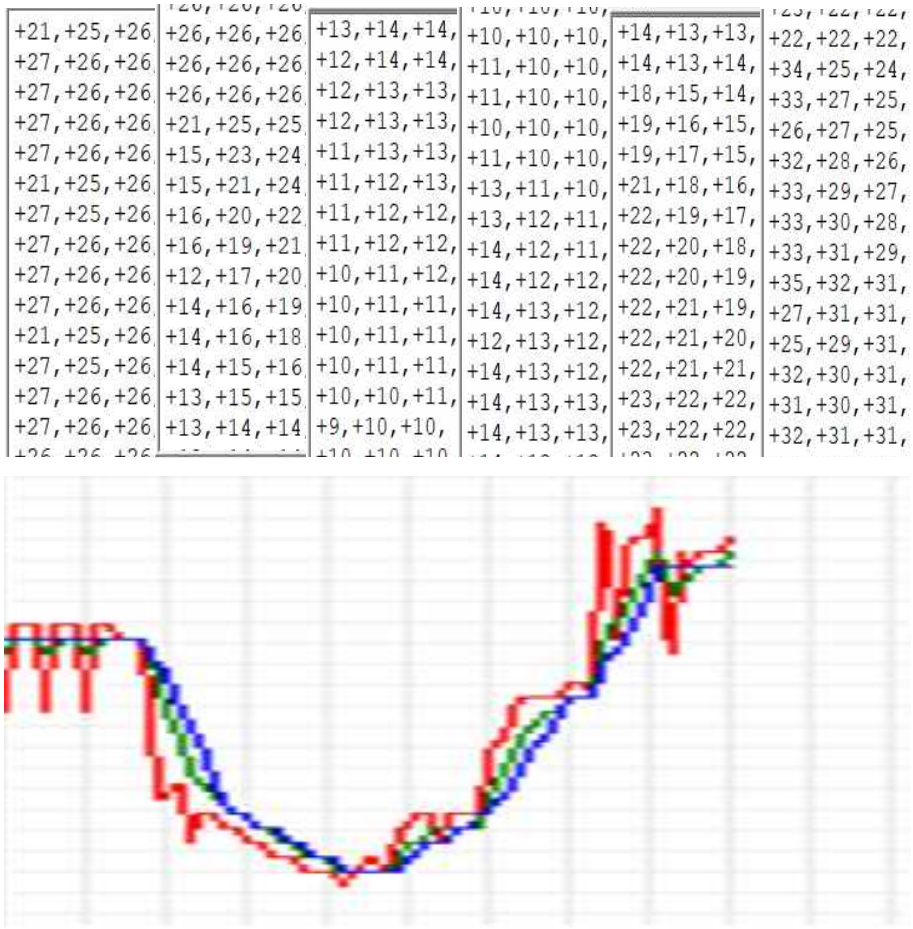
- 데이터시트를 참고하였을 때, PSD 센서가 10~80cm 범위에서는 전압이 거리에 따른 비례방식으로 나타나므로 거리변환 공식을 이용해서 0~5V값을 cm로 환산
- 0~10cm 범위에서 전압이 거리에 따른 비례방식으로 전압이 나타나지 않으므로 예외적으로 처리

```
//PSD
void psd_func(double num)
{
   if((num >= 11)&&(num <= 13))
   {
       set = 1;
       PORTE = (0<<PORTE0)|(0<<PORTE1);
       OCR1B = fast;
   }
   else if(num > 13)
   {
       set = 0;
       PORTE = (0<<PORTE0)|(1<<PORTE1);
       OCR1B = fast;
   }
   else if(num < 11)
   {
       set = 0;
       PORTE = (1<<PORTE0)|(0<<PORTE1);
       OCR1B = fast;
   }
   else
   {
       set = 0;
       PORTE = (0<<PORTE0)|(0<<PORTE1);
       OCR1B = fast;
   }
}
```

- 화분의 위치와 수분 공급원의 위치가 동일할 때만 물이 공급되어야하므로 화분이 적당한 위치에 위치 되었을 때, 전역 변수 set =1로 설정
- PSD와 화분 사이에 거리가 가까울 때는 멀어지도록 모터 방향 설정
- PSD와 화분 사이에 거리가 멀 때는 가까워질 수 있도록 모터 방향 설정
- 모터가 작동될 때, 가변저항을 통해 결정된 속도로 작동할 수 있도록 OCR1B = fast로 설정

<Filtering>

위에서 설명한 Low pass filter와 Moving average filter를 사용해 serial chart로 실제값과 필터값을 받아본 결과 다음과 같았다.



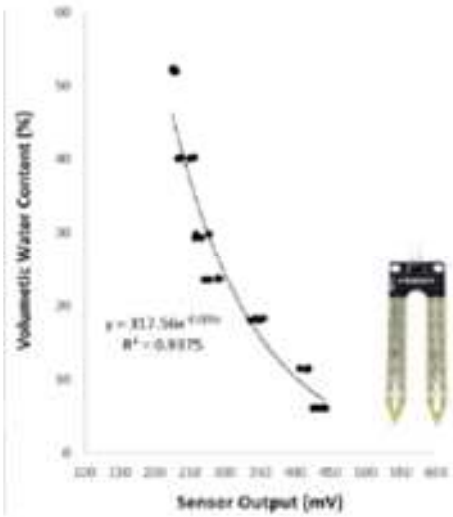
+21,+25,+26,	+26,+26,+26,	+13,+14,+14,	+10,+10,+10,	+14,+13,+13,	+22,+22,+22,
+27,+26,+26,	+26,+26,+26,	+12,+14,+14,	+11,+10,+10,	+14,+13,+14,	+34,+25,+24,
+27,+26,+26,	+26,+26,+26,	+12,+13,+13,	+11,+10,+10,	+18,+15,+14,	+33,+27,+25,
+27,+26,+26,	+21,+25,+25,	+12,+13,+13,	+10,+10,+10,	+19,+16,+15,	+26,+27,+25,
+27,+26,+26,	+15,+23,+24,	+11,+13,+13,	+11,+10,+10,	+19,+17,+15,	+32,+28,+26,
+21,+25,+26,	+15,+21,+24,	+11,+12,+13,	+13,+11,+10,	+21,+18,+16,	+33,+29,+27,
+27,+25,+26,	+16,+20,+22,	+11,+12,+12,	+13,+12,+11,	+22,+19,+17,	+33,+30,+28,
+27,+26,+26,	+16,+19,+21,	+11,+12,+12,	+14,+12,+11,	+22,+20,+18,	+33,+31,+29,
+27,+26,+26,	+12,+17,+20,	+10,+11,+12,	+14,+12,+12,	+22,+20,+19,	+35,+32,+31,
+27,+26,+26,	+14,+16,+19,	+10,+11,+11,	+14,+13,+12,	+22,+21,+19,	+27,+31,+31,
+21,+25,+26,	+14,+16,+18,	+10,+11,+11,	+12,+13,+12,	+22,+21,+20,	+25,+29,+31,
+27,+25,+26,	+14,+15,+16,	+10,+11,+11,	+14,+13,+12,	+22,+21,+21,	+32,+30,+31,
+27,+26,+26,	+13,+15,+15,	+10,+10,+11,	+14,+13,+13,	+23,+22,+22,	+31,+30,+31,
+27,+26,+26,	+13,+14,+14	+9,+10,+10,	+14,+13,+13,	+23,+22,+22,	+32,+31,+31,

< ● 기본값 ● Low pass filter ● moving average filter>

그래프를 확인해본 결과 Low pass filter를 사용하기로 하였다.

▶ 토양 습도 센서

흙이 말랐을 때 식물에게 필요한 물을 자동으로 공급해주기 위해서 토양수분 센서를 사용하기로 결정하였다. 토양수분 센서는 토양 내 수분함량에 따른 저항의 변화를 측정하는 센서인데, 토양 내 수분함량이 많으면 저항값이 작아지고, 수분함량이 적으면 저항값이 커진다.




```
//***토양 습도 센서
double get_MOIST(void)
{
    ADMUX = 0x05;
    ADCSRA |= (1<<ADSC);//0x40;
    while(!(ADCSRA & (1<<ADIF)));
    int adc = ADC;
    double Vadc = adc*5./1023.;

    return Vadc;
}

double MOIST_calculate(double num)
{
    double soil_moist = ((5-num)/5.)*(100);
    return soil_moist; //[per]
}
```

값을 받아오면 위의 사진과 같이 반비례 그래프가 그려지기 때문에 0~1023까지의 ADC값을 0~5V 전압으로 바꿔준 다음 MOIST_calculate에 있는 식으로 토양에 있는 수분을 %로 환산하였다.

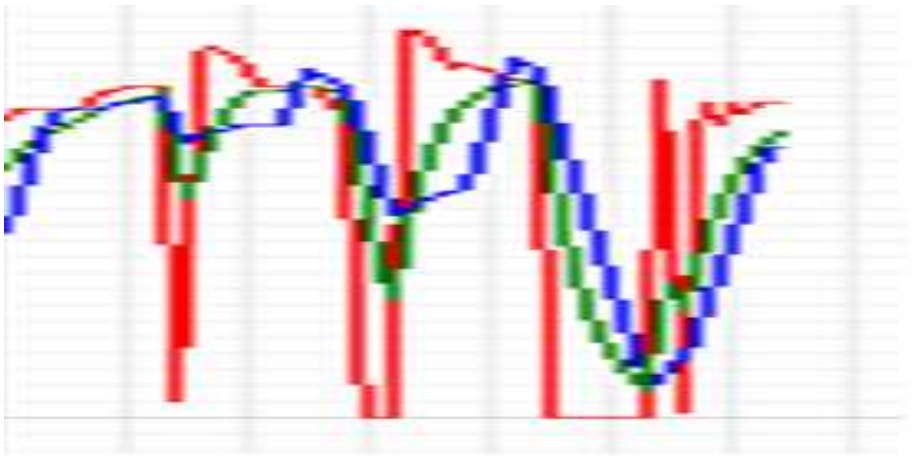
```
//MOIST
void moist_func(double num)
{
    if(num < 60)
    {
        water_on = 1;
    }
    else
    {
        water_on = 0;
    }
}
```

토양의 수분함량이 60%보다 내려가면 식물에 물을 공급하기로 하였고 만약 화분이 다른곳에 위치하였을 때 물이 나오면 안되므로 위치와 토양 수분함량 값이 동시에 맞을 때만 물이 나오게 하기위해 옆의 코드와 같이 water_on이라는 flag 변수를 설정해 주었다.

<Filtering>

위에서 설명한 Low pass filter와 Moving average filter를 사용해 serial chart로 실제값과 필터값을 받아본 결과 다음과 같았다.

	+3,+45,+52,	+12,+47,+54,	+60,+60,+62,	+0,+30,+37,
+53,+44,+33,	+22,+39,+49,	+0,+36,+48,	+0,+45,+56,	+0,+15,+30,
+55,+47,+39,	+65,+45,+50,	+0,+27,+42,	+0,+35,+49,	+0,+12,+24,
+55,+49,+44,	+66,+50,+51,	+0,+21,+36,	+0,+26,+43,	+0,+9,+18,
+55,+50,+50,	+65,+54,+51,	+69,+32,+37,	+0,+20,+37,	+0,+7,+12,
+55,+51,+54,	+64,+56,+52,	+69,+41,+38,	+0,+15,+30,	+0,+5,+6,
+55,+52,+55,	+62,+58,+52,	+67,+47,+39,	+0,+12,+24,	+40,+10,+4,
+55,+53,+55,	+59,+58,+52,	+65,+51,+40,	+0,+9,+18,	+42,+24,+10,
+56,+54,+55,	+59,+58,+52,	+62,+54,+40,	+0,+7,+12,	+1,+19,+10,
+58,+55,+55,	+59,+58,+52,	+63,+56,+41,	+0,+5,+6,	+50,+26,+15,
+58,+56,+56,	+59,+59,+58,	+62,+57,+46,	+60,+18,+6,	+56,+33,+21,
+59,+56,+56,	+59,+59,+62,	+62,+59,+52,	+42,+24,+10,	+52,+30,+26,
+59,+57,+56,	+59,+59,+61,	+61,+59,+58,	+1,+19,+10,	+56,+42,+33,
+59,+57,+57,	+55,+58,+60,	+60,+59,+64,	+50,+26,+15,	+54,+45,+37,
+59,+58,+57,	+59,+58,+59,	+60,+60,+63,		+56,+48,+43,
				+54,+49,+48,
				+56,+51,+48,



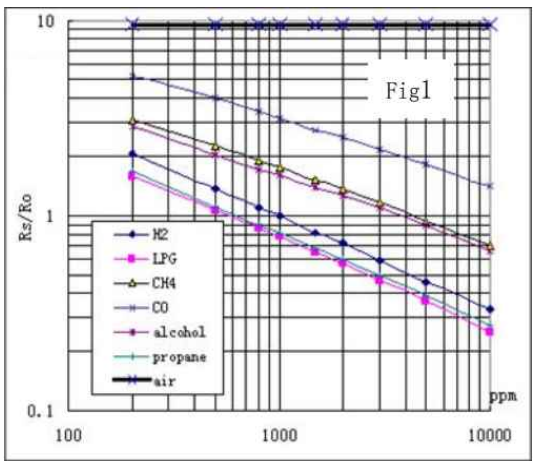
< ● 기본값 ● Low pass filter ● moving average filter>

그래프를 확인해본 결과 Low pass filter를 사용하기로 하였다.

▶ 가스센서(MQ2)

'Smart Planter'안에 담배 연기 등 유해가스가 들어왔을 때 환기를 시켜주기 위해 내부 가스량을 감지하는 센서이다.

MQ2 Datasheet



Gas	a	b
H2	987.99	-2.162
LPG	574.25	-2.222
CO	36974	-3.109
Alcohol	3616.1	-2.675
Propane	658.71	-2.168

Resistance of sensor(Rs): Rs=(Vc/VRL-1)×RL

```
//***가스 센서
double get_GAS(void)
{
    ADMUX = 0x06;
    ADCSRA |= (1<<ADSC);//0x40;
    while(!(ADCSRA & (1<<ADIF)));
    int adc = ADC;
    double Vadc = adc*5./1023.;

    return Vadc;
}
```

- 가스센서를 통해 입력받아 변하는 0~1023 까지의 bit값을 0~5V로 환산하여 출력

```
double GAS_calculate(double num)
{
   double a = 574.25, b=-2.222; //LPG
   double Rs = (5/num-1)*10;
   double ratio = Rs/10;
   double ppm = a*pow(ratio,b);
   return ppm; //[ppm]
}
```

- 0~5V의 값을 GAS_calculate 함수를 통해 ppm값으로 환산
- 가연성 가스인 LPG 감지를 목적으로 하기 위해 데이터시트를 참고해 a = 574.25, b= -2.222로 설정
- Rs = 공기질이 측정되는 저항값 (가변값)
- RL = 칼리브레이션 저항값 (고정값) = 10Ω
- PPM 구하기

ratio = Rs/RL

Result = a * (ratio^b) [ppm]

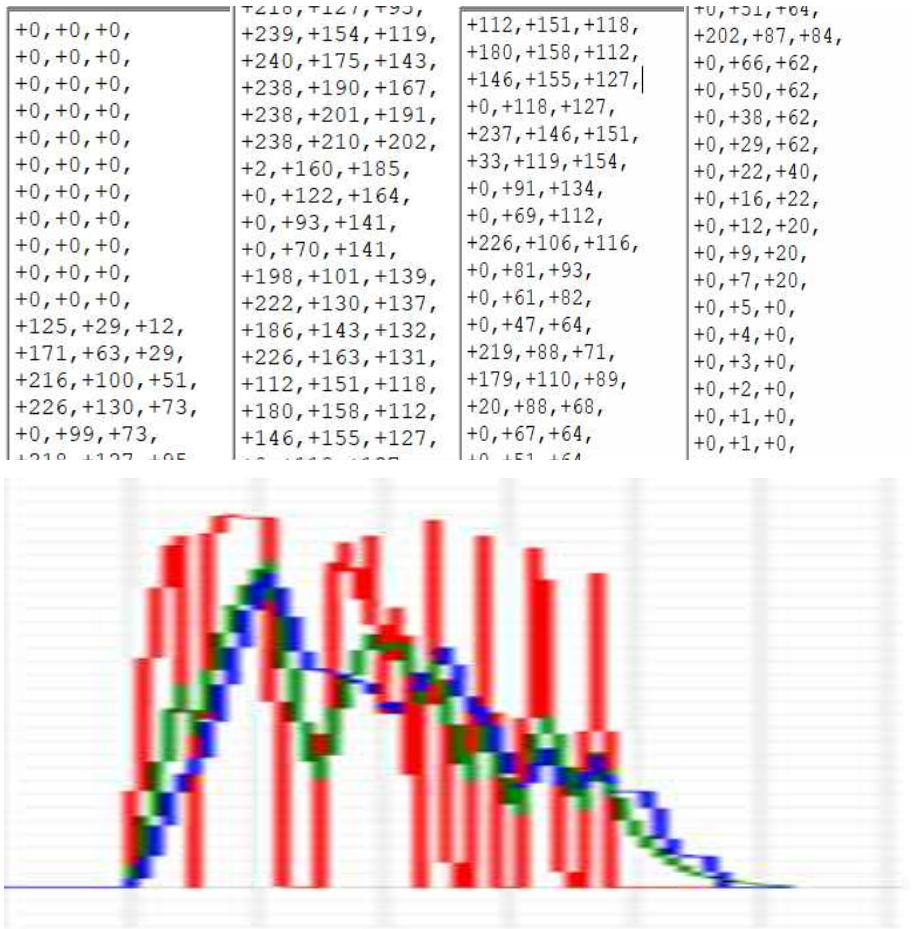
작동 함수

```
//MQ2
void gas_func(double num)
{
    if(num > 150)
    {
       fan_on = 1;
    }
    else
    {
       fan_on = 0;
    }
}
```

- Uart를 통해 Gas 값을 받아본 결과 적절한 값을 설정해 가스가 일정량 이상 감지 됐을 경우 팬이 돌아가게 코딩해주었다.

<Filtering>

위에서 설명한 Low pass filter와 Moving average filter를 사용해 serial chart로 실제값과 필터값을 받아본 결과 다음과 같았다.



< ● 기본값 ● Low pass filter ● moving average filter>

그래프를 확인해본 결과 Low pass filter를 사용하기로 하였다.

4. 제어 주기

```
int main(void)
{
   DDRA = 0xFF;  //led
   DDRB = 0xff;
   DDRC = 0xff;
   DDRE = 0xff;
   DDRF = 0x00;   //adc input pin

    //adc init
   ADMUX = 0x00;   //설정안하면 그냥 0으로됨
   ADCSRA = (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);//=0x87;

   //timer2 interrupt
   TCCR2 = 0x05;
   TCNT2 = 255 - 156;
   TIMSK = (1<<TOIE2);
   sei();

   //Motor PWM
   //B channel clear(compare match) & set(overflow), 14 mode: Fast PWM
   TCCR1A = (1<<COM1B1)|(0<<COM1B0)|(1<<WGM11);
   TCCR1B = (1<<WGM13)|(1<<WGM12)|(1<<CS02)|(0<<CS01)|(1<<CS00);
   //14 mode: Fast PWM, Prescaler 1024
   ICR1 = 312; //Top 값
   OCR1B = 200; //B channel Pwm

   //uart init
   UART1_initialize_polling();

   PORTC = 0x00;
```

- 입력 / 출력을 사용하기 위한 레지스터를 설정
- ADC를 사용하기 위한 레지스터 설정
- Timer1, Timer2를 사용하기 위한 레지스터를 설정
- Uart 사용하기 위한 레지스터 설정

<Timer2 제어주기>

센서값을 받고 선형화 및 수치화하는 작업은 2번 타이머카운터의 제어주기 내에서 작동한다. 다른 타이머카운터 인터럽트를 쓰게 될 경우에 대비하여 순위가 비교적 높은 2번 타이머 카운터를 선택하였으나 결론적으로 다른 타이머카운터 인터럽트는 사용하지 않았다.

2번 타이머카운터의 제어주기는 10ms로 사용하였고 주기를 만든 후 인터럽트가 100번 실행될 때마다, 즉 1초마다 한번만 구문들을 실행하도록 하였다.

<Timer1 제어주기>

모터의 속도를 조절하기 위해 모터 드라이브(L298N)을 사용하였고, 모터가 작동하는 B channel을 compare match mode, Fast PWM mode로 설정하였다. 20ms 주기의 PWM 생성하기 위해 prescaler을 1024로 설정하여 ICR1 값을 구하고 OCR1 값을 이용해서 Duty Ratio 설정하였다.

```
while (1)
{
  //0b(0)(0)(펌프-)(펌프+)(0)(0)(led)(fan)
  if((!fan_on)&&(!led_on)&&(!water_on)&&(!set))
  {
     PORTC = 0b00000000;
  }
   else if((fan_on)&&(!led_on)&&(!water_on)&&(!set))
   {
       PORTC = 0b00000001;
  }
  else if((!fan_on)&&(led_on)&&(!water_on)&&(!set))
  {
     PORTC = 0b00000010;
  }
  else if((!fan_on)&&(!led_on)&&(water_on)&&(!set))
  {
     PORTC = 0b00000000;
  }
  else if((!fan_on)&&(!led_on)&&(!water_on)&&(set))
  {
     PORTC = 0b00000000;
  }
  else if((fan_on)&&(led_on)&&(!water_on)&&(!set))
  {
     PORTC = 0b00000011;
  }
  else if((fan_on)&&(!led_on)&&(water_on)&&(!set))
  {
     PORTC = 0b00000001;
  }
  else if((fan_on)&&(!led_on)&&(!water_on)&&(set))
  {
     PORTC = 0b00000001;
  }
  else if((!fan_on)&&(led_on)&&(water_on)&&(!set))
  {
     PORTC = 0b00000010;
  }
  else if((!fan_on)&&(led_on)&&(!water_on)&&(set))
  {
     PORTC = 0b00000010;
  }
  else if((!fan_on)&&(!led_on)&&(water_on)&&(set))
  {
     PORTC = 0b00010000;
  }
  else if((fan_on)&&(led_on)&&(water_on)&&(!set))
  {
     PORTC = 0b00000011;
  }
  else if((fan_on)&&(led_on)&&(!water_on)&&(set))
  {
     PORTC = 0b00000011;
  }
  else if((fan_on)&&(!led_on)&&(water_on)&&(set))
  {
     PORTC = 0b00010001;
  }
  else if((!fan_on)&&(led_on)&&(water_on)&&(set))
  {
     PORTC = 0b00010010;
  }

  else if((fan_on)&&(led_on)&&(water_on)&&(set))
  {
     PORTC = 0b00010011;
  }
  else
  {
     PORTC =0b00000000;
  }
}
```

Timer2에서 PORTC를 변경해주면 어두울 때 천장 LED가 계속 켜있는게 아니고 주기마다 깜박거리는 문제점이 발생하여서 관련 함수는 전역변수 flag 값을 변경해주고 Timer2와 별개로 while문 내부에서 상황에 따른 flag 변수에 따라 작동할 수 있도록 하였다.

센서를 하나씩 확인해 볼 때는 정상 작동하였지만 하나로 합쳐서 진행할 때 천장 LED / 시스템 팬 / 수중 펌프를 같은 PORTC에 연결해뒀기 때문에 두 가지 이상의 상황이 동시에 이루어지면 서로 충돌나는 경우가 있었다. 이에 해결방안으로 16가지 경우로 나눠 각각의 경우에 맞게 PORT를 설정하였다.

5. 진행 방법

한준호	황유환
센서 선형화 및 수식화	
나무틀 제조	아크릴 제조
가스 센서, psd 센서, lm35, 가변저항 파트 보고서 제작 및 보고서 통합	서론, 하드웨어 구성, 토양수분 센서, CDS 센서 파트 보고서 제작

주당 2~3회 정도 만나면서 하드웨어 제작 및 아이디어 회의를 진행. 센서 선형화 및 수식화를 어떤 방식으로 할지 충분히 논의한 후 과제형식으로 코드를 만들어온 뒤 작동방식을 비교해보고 더 좋은 코드를 사용하는 방식으로 진행.

6. 고찰

Timer2에서 PORTC를 변경해주면 주기마다 어두울 때 천장 LED가 계속 켜 있는게 아니고 계속 꺼져서 깜박거리는 문제점이 발생하여서 관련 함수는 전역변수 flag 값을 변경해주고 Timer2와 별개로 while문 내부에서 상황에 따른 flag 변수에 따라 작동할 수 있도록 하였다.

센서를 하나씩 확인해 볼 때는 정상 작동하였지만 하나로 합쳐서 진행할 때 천장 LED / 시스템 팬 / 수중 펌프를 같은 PORTC에 연결해뒀기 때문에 두 가지 이상의 상황이 동시에 이루어질 때 서로 충돌나는 경우가 있었다. 이에 해결방안으로 16가지 경우로 나눠 각각의 경우에 맞게 PORT를 설정하였다.

작품을 진행하며 실험실 파워 서플라이와 개인이 소장한 파워 서플라이를 썼을 때 같은 크기의 전압을 인가하여도 모터의 출력이 다른점을 확인할 수 있었다. 이에 프로젝트를 진행할 때 실험실의 파워 서플라이로 진행을 택함.

LPF와 MAV필터를 사용해 필터링을 진행한 결과 본래값은 값을 급격히 변화시킬 때 바로바로 따라오지만 LPF, MAV필터 순서대로 delay가 조금씩 생기는걸 확인하였다. 필터링을 사용하면 값이 튀었을 때 조금더 안정적이고 연속적으로 센서값을 받을 수 있지만 delay가 생겨 과연 필터링을 사용하는 것이 좋은가? 라는 생각을 하였다. 이는 요즘 나오는 센서들은 기본적으로 필터링이 내재 되어있기에 발생하는 현상이라는 결론을 내렸다.