로봇학 실험 3 결과보고서

사람의 수면을 돕는 스마트 침실

2021741062 김현준

2021741074 염준우

서론

개발동기

충분한 수면시간은 인간에게 필수적이지만, 그러지 못하는 게 현실이다. 그렇기에 단시간 잠을 자더라도 숙면을 취하려면 어떠한 환경이 필요한지 고민하게 되었다. 스마트 침실의 역할을 수행함과 동시에 스마트 홈의 역할도 수행할 수 있도록 작품을 구상하게 되었다.

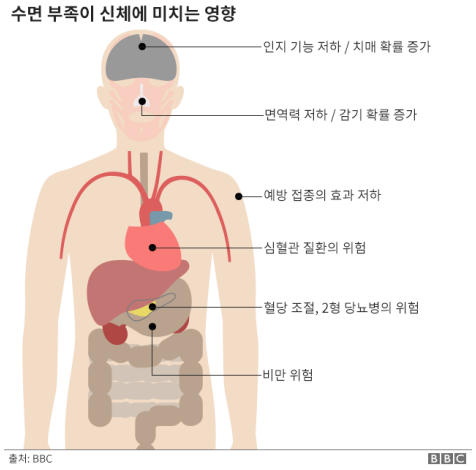


그림1. BBC가 발표한 수면 부족이 신체에 미치는 영향

https://www.bbc.com/korean/features-41775870

시뮬레이션 과정

작품의 시뮬레이션은 1층에 있는 스위치를 누르는 것부터 시작한다. 스위치를 누르면 1층에 있는 LED가 점점 밝아진다. 태양의 역할을 하는 LED가 밝아지면 조도센서에서 받은 LUX 값을 바탕으로 시스템이 현재 낮인지 밤인지를 구분하게 된다. 낮/밤에 따라 서보 모터와 부저가 작동하며, 시간대와 상관없이 습도센서, 온도센서, 가속도 센서, 자이로 센서가 작동한다.

낮에는 환기를 위해 서보 모터가 작동하여 문이 열리고 잠에 든 사용자를 깨우기 위해 부저가 작동한다. 밤에는 소음 차단을 위해 열려 있던 문이 자동으로 닫히게 된다.

습도센서, 온도 센서는 사용자에게 현재 상태를 나타내기 위해 사용하였다. 온도 센서는 2층에 마련된 자동온도제어기와 연결되어 있다. 자동온도제어기는 현재 온도로 저장된 온도보다 주위 온도가 높아지면 온도를 내리기 위해 냉방기를 작동시키게 하고, 주위 온도를 현재 온도로 업데이

트한다. 주위 온도가 현재 온도보다 차가워지면 난방기를 작동시키게 한다. 습도 센서는 7 세그먼트와 연결되어 있고 측정된 습도 센서 값과 온도센서 값을 바탕으로 불쾌지수를 1~9로 환산하여 사용자에게 보여준다.

초음파 센서는 화장대에 설치되어 있는데, 사용자가 일어난 후 화장대 앞으로 이동할 때 특정 거리안에 들어오면 자동으로 LED를 점등하여 사용자가 편안하게 외출 준비를 할 수 있도록 하였다.

가속도, 자이로 센서는 침대에 설치되어 있다. 최근 다양한 업체들이 침대에 센서를 부착하고 센서 값을 바탕으로 사용자의 수면 점수를 분석하는 부분에서 채용한 부분이다. (밑의 그림은 레스피오 라는 업체이다.)가속도, 자이로 센서들의 값과 수면의 질 사이의 관계를 파악하지 못하여 얻은 값을 더 큰 값으로 가공하지는 못했지만, 침대가 움직이며 바뀌는 가속도, 자이로 센서 값을 PC 쪽으로 일정 시간 마다 전달하게 된다.

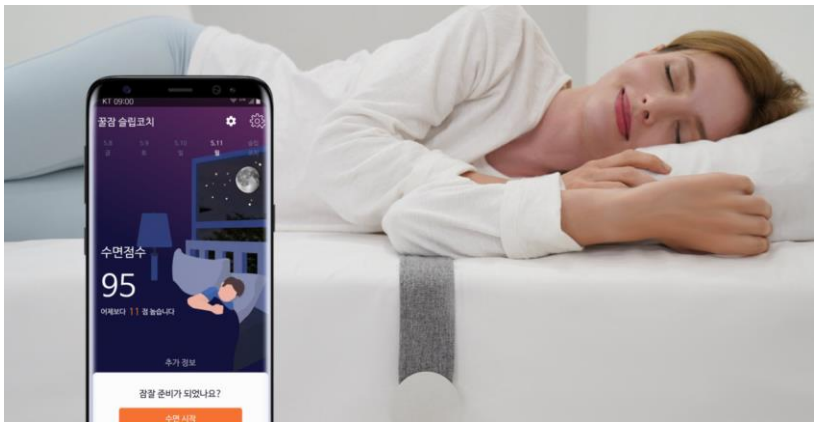
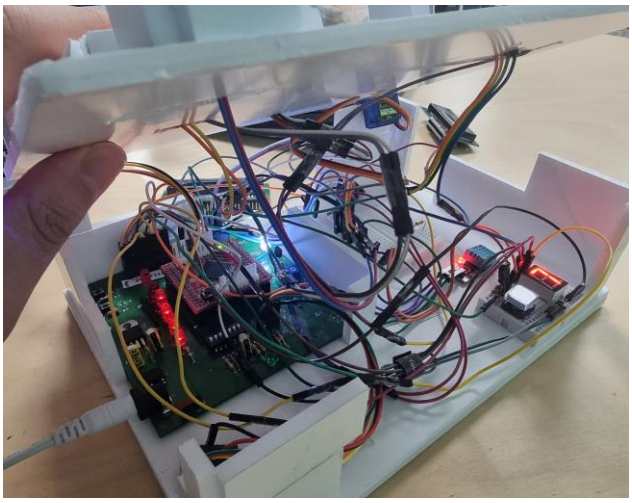


그림2. 수면 분석 업체 중 하나인 레스피오

https://www.vitealth.net/

2. 하드웨어 설명

하드웨어는 포맥스를 이용하여 제작하였으며 1층과 2층으로 나뉜다. 2층 부분이 스마트 침실 부분이고, 1층 부분은 스마트 침실을 위한 회로 부분이다. 스마트 침실에는 초음파, 온도, 습도, 가속도, 자이로, 조도 센서가 사용되었다. 센서들은 스마트 침실의 가구에 배치하여 사용자와 상호작용할 수 있도록 구성하였다. 침실의 동작을 위한 부품으로는 서보 모터, 3색 LED, LED, 스위치, 7 세그먼트, 부저가 사용되었다.

3. 시스템 설명

센서에 사용한 필터들

MAF와 LPF를 사용하였다. MAF는 일정 시간 동안의 DATA를 평균내서 사용하기 때문에 전반적인 신호를 평탄화 할 수 있다. 하지만 MAF는 이동 평균을 계산하기 때문에 실시간성을 요구하는 프로그램과 고주파 필터링 작업에는 적합하지 않다. 그렇기에 LPF 필터를 같이 사용하여 MAF 필터의 단점을 보완하고자 했다. LPF 필터의 시간 차이는 필터가 다시 실행되는 시간을 추정하여 대략 0.5초에 한번 실행된다고 가정해서 시간간격을 0.5로 잡았다. 조금 더 정확하게 하고 싶다면 Timer counter를 이용해 시작부터 끝까지 Timestamp를 찍어서 하는 방법이 있지만 이 경우에 오랜 시간동안 작품을 실행시킬 경우 OverFlow가 발생할 가능성이 있어서 이렇게 하지는 않았다. Overflow를 방지하는 방법이 있긴 하지만 이 방법도 문제가 있어 작품의 안정성을 위해서는 이렇게 시간 간격을 고정하는 방식이 제일 낫다고 판단했다

센서들에 따라 MAF와 LPF 중 한 개의 필터를 사용하여 센서 노이즈를 감소시키고자 하였다.

LM35, DHT11은 온/습도 센서로써 온/습도가 갑자기 크게 바뀌는 것이 아니기에 센서 값의 전반적인 평탄화가 더 적합하다고 생각하여 MAF 필터를 적용하였다.

```
// 필터 통과
// mafp = Moving Average Filter Passed

double mafp_temperature = maf_lm35.Weight_MAF(temperature);
double lpfp_lux = lpf_cds.digitalLPF(lux);
double mafp_humidity = maf_dht11_hum.Weight_MAF(humidity);

double lpfp_distance_mm = lpf_ultrasonic_sensor.digitalLPF(distance_mm);

double lpfp_a_x = lpf_a_x.digitalLPF(a_x);
double lpfp_a_y = lpf_a_x.digitalLPF(a_y);
double lpfp_a_z = lpf_a_x.digitalLPF(a_z);

double lpfp_g_x = lpf_a_x.digitalLPF(g_x);
double lpfp_g_y = lpf_a_x.digitalLPF(g_y);
double lpfp_g_z = lpf_a_x.digitalLPF(g_z);
```

```cpp
#include "MAF_Filter.h"

// default constructor
MAF_Filter::MAF_Filter()
{
    for (int i = 0; i < MAF_SIZE; ++i)
    {
        Weight_data_list[i] = 0.0;
    }
    Weight_avg_count_flag = 1;
    Weight_MAF_count = 0;
} //MAF_Filter

// default destructor
MAF_Filter::~MAF_Filter()
{
} //~MAF_Filter
// 무게센서 MAF 필터
double MAF_Filter::Weight_MAF(double Weight_data){
    // 평균필터 값 집어넣기
    Weight_data_list[Weight_MAF_count++] = Weight_data;
    if (Weight_MAF_count >= MAF_SIZE)
    {
        Weight_MAF_count = 0;
        Weight_avg_count_flag = 0;
    }
    // 평균필터 합 구하기
    double MAF_Sum = 0;
    for(int i = 0; i < MAF_SIZE; i++){
        MAF_Sum += Weight_data_list[i];
    }
    if(Weight_avg_count_flag)
    {
        return MAF_Sum / Weight_MAF_count;
    }
    return MAF_Sum / MAF_SIZE;
}
```

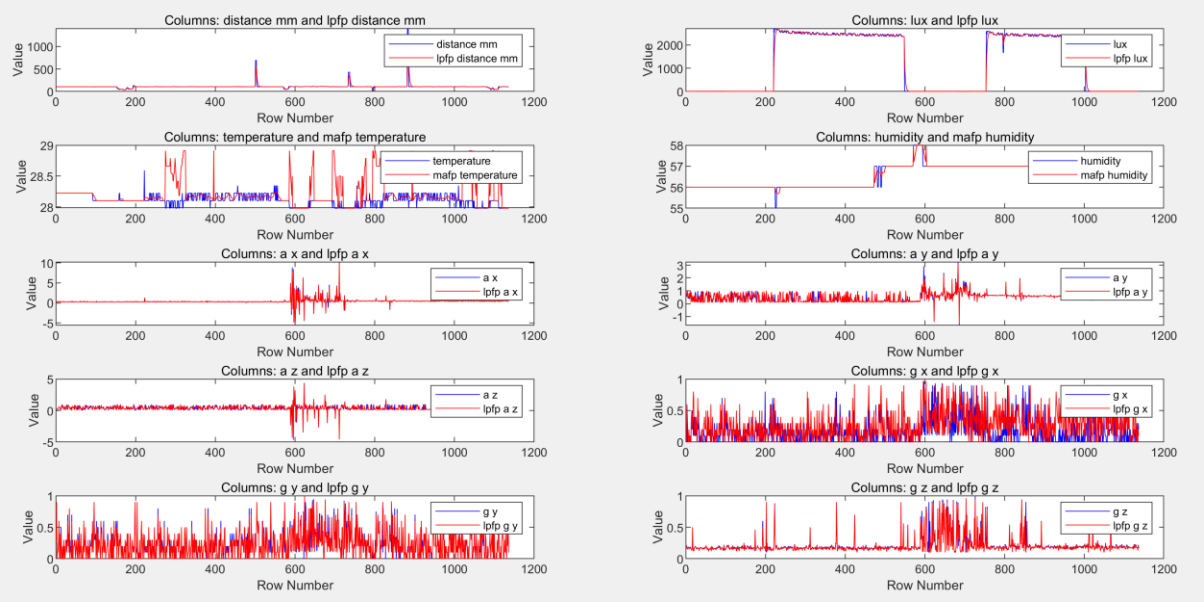
```cpp
#include "LPF_Filter.h"

// default constructor
LPF_Filter::LPF_Filter(double Fcutoff, double DeltaT) : Fcut(Fcutoff), deltaT(DeltaT)
{
    //this->Fcut = Fcut;
    this->LPF_prev_value = -10000; // 값이 정해지지 않았을때 초기 값
    //this->deltaT = 0.5;
} //LPF_Filter

// default destructor
LPF_Filter::~LPF_Filter()
{
} //~LPF_Filter
double LPF_Filter::digitalLPF(double value)
{
    double tau = 1/(2*PI*Fcut);


    if (this->LPF_prev_value == -10000)
    {
        this->LPF_prev_value = value;
        return this->LPF_prev_value;
    }
    else
    {
        double filtered_value;
        filtered_value = (deltaT * value + tau * LPF_prev_value)/(tau + deltaT);
        LPF_prev_value = filtered_value;
        return filtered_value;
    }
}
```

LPF 계산 공식

```
double calcLPF(p_X, p_pre_Y, p_Fc, p_dt)
{                    Input   이전 Output  차단 주파수  제어 주기
```

$$\therefore v_o(t) = \frac{((dt)v_i(t) + (\tau)v_o(t-1))}{(\tau + dt)}$$

```
}
```



이 그래프는 매트랩으로 MAF, LPF를 적용시킨 결과이다. Colums1~10은 필터링하기 전의 값이고 Colums 11~20은 필터링 된 값이다.

순서대로 distance_mm. lux, temperature, humidity, a_x, a_y, a_z, g_x, g_y, g_z 를 의미한다.

Temperature 부분이 많이 오차가 심한 것 처럼 보이지만 y축을 확인해본 결과 1도 범위 내에서 움직이고 있음을 확인하였고, 센서 자체의 오차라고 분석하였다.

아래 코드는 UART 통신을 통해 센서 값을 받아오는 부분이다.

```
USART.USART0_TX_array_funct(distance_mm);
USART.USART0_TX_array_funct(",");
USART.USART0_TX_array_funct(lux);
USART.USART0_TX_array_funct(",");
USART.USART0_TX_array_funct(temperature);
USART.USART0_TX_array_funct(",");
USART.USART0_TX_array_funct(humidity);
USART.USART0_TX_array_funct(",");
USART.USART0_TX_array_funct(a_x);
USART.USART0_TX_array_funct(",");
USART.USART0_TX_array_funct(a_y);
USART.USART0_TX_array_funct(",");
USART.USART0_TX_array_funct(a_z);
USART.USART0_TX_array_funct(",");
USART.USART0_TX_array_funct(g_x);
USART.USART0_TX_array_funct(",");
USART.USART0_TX_array_funct(g_y);
USART.USART0_TX_array_funct(",");
USART.USART0_TX_array_funct(g_z);
USART.USART0_TX_array_funct(",");
```

```
USART.USART0_TX_array_funct(lpfp_distance_mm);
USART.USART0_TX_array_funct(",");
USART.USART0_TX_array_funct(lpfp_lux);
USART.USART0_TX_array_funct(",");
USART.USART0_TX_array_funct(mafp_temperature);
USART.USART0_TX_array_funct(",");
USART.USART0_TX_array_funct(mafp_humidity);
USART.USART0_TX_array_funct(",");
USART.USART0_TX_array_funct(lpfp_a_x);
USART.USART0_TX_array_funct(",");
USART.USART0_TX_array_funct(lpfp_a_y);
USART.USART0_TX_array_funct(",");
USART.USART0_TX_array_funct(lpfp_a_z);
USART.USART0_TX_array_funct(",");
USART.USART0_TX_array_funct(lpfp_g_x);
USART.USART0_TX_array_funct(",");
USART.USART0_TX_array_funct(lpfp_g_y);
USART.USART0_TX_array_funct(",");
USART.USART0_TX_array_funct(lpfp_g_z);
USART.USART0_TX_endl();
```

조도 센서

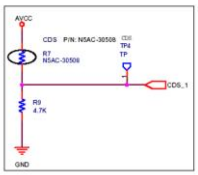
조도 센서는 제공받은 GR5539를 사용하였다. 1층에 있는 스위치를 눌러 인공 태양(LED)의 밝기가 밝아지면 이를 센서 값으로 받아, 일정 경계가 넘어가면 시스템에 아침이라는 정보를 제공한다. CDS에서 받은 값을 그대로 사용하는 것이 아닌 LUX 값으로 환산하여 사용하였다. ADC를 사용하여 값을 구하였고, 조도센서, 온도센서 등과 같이 ADC를 활용해서 값을 구하는 센서들에 쉽게 접근하기 위해 ADC_Control이라는 헤더파일을 만들어 사용하였다. 사용한 공식은 수업시간에 배운 것과 동일하다. 코드에서는 LUX 값이 100 이상이면 아침으로 인식하고 서보 모터와 부저를 울리도록 설계하였다.

```
double ADC_Control::cal_cds(double Vout)
{

   // GL5539 CDS cell 의 gamma R100 기준 10~15K Ohm -> 12.5 K Ohm으로 평균
   const double Gamma = 0.5051499783;
   const double R9 = 4700;
   const double AVCC = ADC_Control::Vadc;

   double Rcds = (R9 * AVCC)/Vout - R9;

   double Lux = pow(10, 1-(log10(Rcds)-log10(40000))/Gamma);

   return Lux;

}
```

```
void ActionMorning(int lux, int day_std_lux)
{
    if (lux > day_std_lux)
    {
        servo_rotate(100);
        alarm = 1;
    }
    else
    {
        servo_rotate(0);
        alarm = 0;
    }
}
```

◆ CDS

· ADC Data 수치화



▪ Find R_{cds}

$$V_{out} = \frac{R_9}{R_9 + R_{cds}} \times AVCC$$

$$\therefore R_{cds} = \frac{R_9 \times AVCC}{V_{out}} - R_9$$

▪ Find Gamma

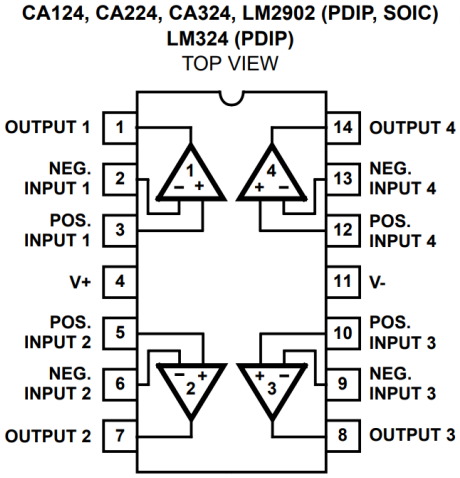
$$\therefore \gamma = \frac{log(R_{cds}\ [\Omega]) - log(40\ [k\Omega])}{log(10\ [Lux]) - log(x\ [Lux])}$$

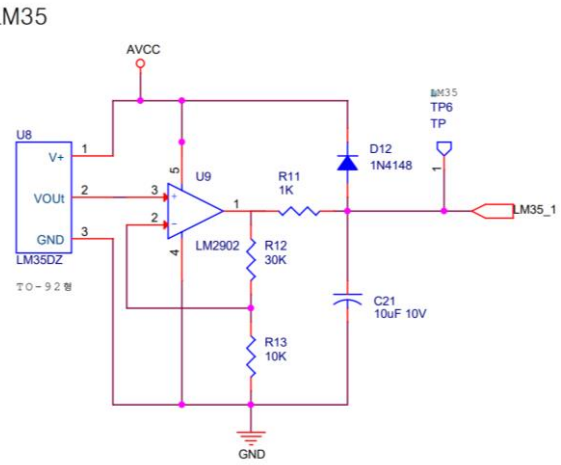
▪ Find Lux

$$log(x\ [Lux]) = 1 - \frac{log(R_{cds}\ [\Omega]) - log(40\ [k\Omega])}{\gamma}$$

$$\therefore x\ [Lux] = 10^{1 - \frac{log(R_{cds}\ [\Omega]) - log(40\ [k\Omega])}{\gamma}}$$

온도 센서





온도 센서는 주어진 LM35를 사용했다. LM35는 원래의 회로환경에서는 사용할 수 없었지만 기존에 달려 있는 증폭기 회로를 고쳐 사용이 가능하게 만들었다. 왼쪽의 LM2902 데이터 시트를 살펴보면 OpAmp의 입출력 부분은 알맞게 연결되어 있지만, 증폭기의 전원부가 잘못 연결된 것을 알 수 있다. DataSheet에는 4번 11번이 V+, V- 이지만 실습회로에서는 그 반대로 되어있다. 외부 보드를 따로 만들어서 핀을 새로 연결하여 사용하였다. 현재 증폭기가 4배 증폭을 하는 중이기 때문에 10mv당 1도가 올라가던 LM35는 40mv당 1도가 올라가도록 바뀌었다.

이후 LM35를 활용하여 온도를 자동으로 조절하는 온도 조절기를 만들었다. 기준 온도보다 외부의 온도가 높으면 빨간색 LED를 점등하여 냉방기를 가동하고 기준 온도를 외부의 온도와 동기화 시킨다. 만약 기준 온도보다 외부의 온도가 낮으면 파란색 LED를 점등하여 난방기를 가동하고 온도를 업데이트 한다.

```
if (dif_temp > offset_temp) // 기준 온도 보다 높다.
{
    USART.USART0_TX_array_funct("hot");

    PORTA |= 0x02; // 빨간색 LED
    PORTA &= ~0x01; // 파란색 LED
    PORTA &= ~0x04; // 초록색 LED
    // 빨간색 1번만 키고 나머지 LED는 끔 그 외에 포트는 건들지 않음
}
else if (abs(dif_temp) < offset_temp) // 온도가 현재 온도랑 비슷하다.
{
    PORTA |= 0x04; // 초록색 LED
    PORTA &= ~0x01; // 파란색 LED
    PORTA &= ~0x02; // 빨간색 LED
    // 초록색 2번만 키고 나머지 LED는 끔 그 외에 포트는 건들지 않음
    USART.USART0_TX_array_funct("std");
}
else // 기준 온도 보다 더 낮다.
{
    PORTA |= 0x01; // 파란색 LED
    PORTA &= ~0x02; // 빨간색 LED
    PORTA &= ~0x04; // 초록색 LED

    USART.USART0_TX_array_funct("cool");
    // 파란색만 키고 나머지 LED는 끔 그 외에 포트는 건들지 않음
}
```

습도 센서

습도 센서는 DHT11 온습도 센서에서 습도 값 만을 사용하였다. 자체적인 통신 규약이 있어서 이에 따라 코드를 구현했다. ADC_CONTROL에서는 구현하지 못하기에 DHT11 전용 헤더파일을 만들어 진행하였다. DHT11을 읽는 동안에는 잠시 인터럽트를 중지하는데 그 이유는 다른 인터럽트가 DHT11의 타이밍을 방해하기 때문에 DHT11의 올바른 값을 받아올 수 없기 때문이다. 다른 인터럽트를 중지하기 때문에 이 여파로 부저가 울렸을 때 부저의 음이 지연되어 일정한 주기의 부저의 음을 들을 수 없게 된다.

습도센서 값을 받아오면 우리가 알고 있는 상대습도의 개념으로 값을 받아오게 된다. 받아온 값으로 무엇을 하면 사용자에게 더 쓸모 있는 정보가 될 수 있을지 고민하다 앞에서 LM35를 활용해 얻은 온도센서 값과 결합하여 불쾌지수를 계산하고자 하였다. 불쾌지수는 다음과 같은 식으로 계산할 수 있다.

$$0.81 * \text{섭씨온도} + 0.01 * \text{상대습도}(\%) (0.99 * \text{섭씨온도} - 14.3) + 46.3$$

계산한 불쾌지수는 7 – 세그먼트를 이용하여 십의 자릿수만 표현하여 1~9 단계로 정보를 제공할 수 있도록 하였다.

```
int8_t DHT11::readDHT11(uint8_t* temperature, uint8_t* humidity) {

    //다른 인터럽트가 타이밍에 방해를 해서 dht11을 읽는 동안에는 잠시 인터럽트를 종료
    uint8_t bits[5] = {0};
    uint8_t cnt = 7;
    uint8_t idx = 0;

    // 시작 신호 전송
    DDRF |= (1 << DHT11_PIN);  // 출력 모드로 설정
    PORTF &= ~(1 << DHT11_PIN);  // 0으로 설정
    cli();
    _delay_ms(20);  // 최소 18ms 이상 대기
    PORTF |= (1 << DHT11_PIN);  // 1로 설정
    _delay_us(1);
    DDRF &= ~(1 << DHT11_PIN);  // 입력 모드로 변경
    _delay_us(39);

    // 센서로부터 응답 확인
    if ((PINF & (1 << DHT11_PIN))) {
        return -1;  // 응답 없음
    }
    _delay_us(80);
    if (!(PINF & (1 << DHT11_PIN))) {
        return -1;  // 응답 없음
    }
    _delay_us(80);

    // 데이터 수신
    for (uint8_t i = 0; i < 5; i++) {
        for (uint8_t j = 0; j < 8; j++) {
            while (!(PINF & (1 << DHT11_PIN)));  // 비트 시작을 기다림
            _delay_us(40);
            if (PINF & (1 << DHT11_PIN)) {
                bits[i] |= (1 << cnt);  // 1이면 비트에 값을 설정
            }
            while (PINF & (1 << DHT11_PIN));  // 비트의 끝을 기다림
            cnt--;
        }
        cnt = 7;
    }
    sei();
```

자이로, 가속도 센서

가속도, 자이로 센서는 MPU6050 칩을 사용하였으며 가속도와 각속도의 값을 같이 받을 수 있게 하였다. 실습 보드와 I2C 통신을 하여 값을 받아왔다. 가속도 센서의 단위는 m/s^2, 자이로 센서의 단위는 rad/s 로 단위를 환산하여 시리얼 모니터에 표현하도록 만들었다.

```
double Gyro::get_a_x()
{

    a_x = (a_x_h<<8) | a_x_l;
    double calibrated = a_x - bas_a_x;
    a_x = (calibrated/AFS_SEL)*9.81; // m/s^2 단위로 변경
    return a_x;
}

double Gyro::get_a_y()
{
    a_y = (a_y_h<<8) | a_y_l;
    double calibrated = a_y - bas_a_y;
    a_y = (calibrated/AFS_SEL)*9.81; // m/s^2 단위로 변경
    return a_y;
}

double Gyro::get_a_z()
{
    a_z = (a_z_h<<8) | a_z_l;
    double calibrated = a_z - bas_a_z;
    a_z = (calibrated/AFS_SEL)*9.81; // m/s^2 단위로 변경
    return a_z;
}
```

```
double Gyro::get_g_x()
{
    g_x = (g_x_h<<8) | g_x_l;
    g_x = (g_x - bas_g_x)/FS_SEL;
    return deg2rad(g_x);
}

double Gyro::get_g_y()
{
    g_y = (g_y_h<<8) | g_y_l;
    g_y = (g_y - bas_g_y)/FS_SEL;
    return deg2rad(g_y);
}

double Gyro::get_g_z()
{
    g_z = (g_z_h<<8) | g_z_l;
    g_z = (g_z - bas_g_z)/FS_SEL;
    return deg2rad(g_z);
}
```

초음파 센서

초음파 센서는 HC-SR04 모듈을 사용했다. 초음파 센서는 VCC, GND 말고도 Trigger 핀과 Echo 핀을 이용해 물체와의 거리를 측정한다. 음파가 다시 돌아오는 시점을 구하여 시간 차이를 계산하고 소리의 속도를 곱하여 거리를 계산한다. 초음파 센서는 화장대에 설치하여 사용자가 가까이 가면 자동으로 전등이 들어올 수 있도록 구성하였다.

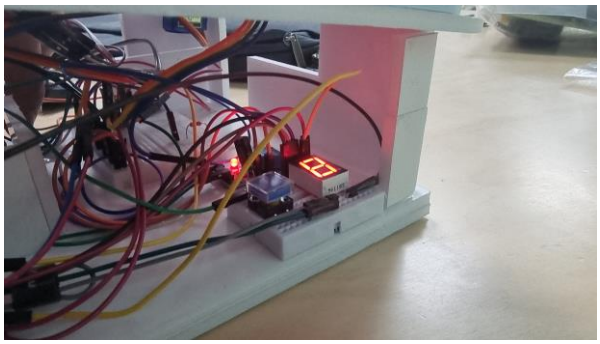
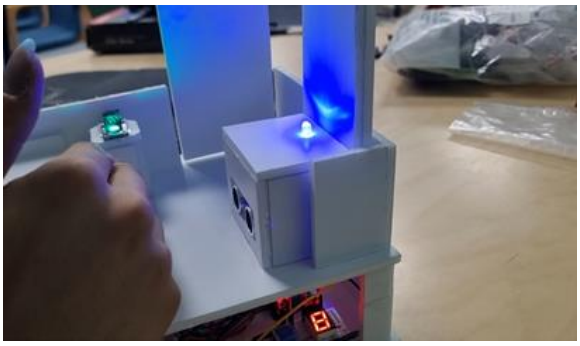
```
void sonic_init()
{
    TCNT1 = 0;//65526;
    TCCR1B |= 0xC2; //6: 1 상승 엣지 [2:0]:010 Prescaler 8,노이즈 제거
    DDRB |= 0x40;// PB6 trigger핀 출력 활성화, PB5 LED 활성화
    TIMSK |= 0x25;
    //bit5: timer counter 1 input caputre,
    //bit2:overflow interrupt 1
    //bit0:overflow interrupt 0
    //TIMER0
    TCNT0 |= 246;//TCNT0 246 시작값
    TCCR0 = 0x02;//Prescaler 8
}
```

계산 공식

```
distance_mm = round(((dif_time)*(8./F_CPU)*velocity/2.)*1000.);
```

시뮬레이션 결과

1. 스위치를 누르면 Fade_led에 의해 led가 서서히 켜진다. 스위치를 다시 누르면 LED가 서서히 꺼지게 된다.
2. Led가 켜지면 CDS에서 센서 값을 ADC 받아 LUX로 변환하고, 100 LUX 이상이 되면 아침 상황이 되고, 서보모터와 부저 관련 코드를 실행한다. 문이 열리는 연출을 위해 서보모터가 90도 회전하고, 알람 역할을 하는 부저가 작동한다.
3. 온도 센서, 습도 센서가 작동하며 온도가 바뀜에 따라 온도 조절 장치가 실행되고 습도 센서 값과 함께 불쾌 지수를 계산하여 1층에 있는 7-세그먼트에 십의자리 숫자로 나타낸다.
4. 화장대에 사람이 다가가면 일정 거리 미만일 때 LED를 점등한다.

4. 고찰

우리 조는 다양한 센서를 활용하여 스마트 침실을 개발하였다. 처음계획과는 많이 달라진 부분도 있었지만 큰 틀을 유지하며 좋은 결과물을 만들었다고 생각한다.

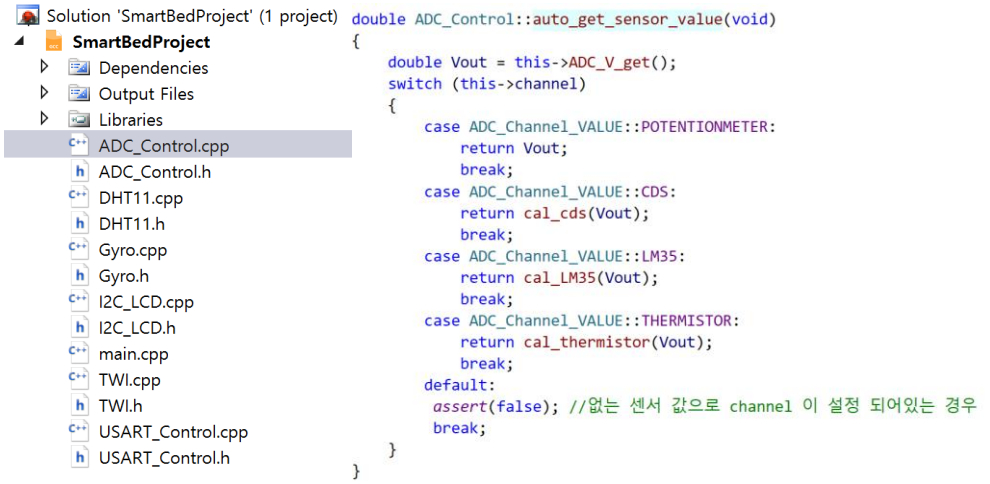
우리 조는 각자 자신 있는 분야를 맡아 분업을 하였는데, 김현준은 알고리즘 설계와 하드웨어 설계 및 제작을, 염준우는 소프트웨어 프로그래밍 및 회로 설계를 맡았다. 두 명이 같이 하드웨어를 제작하거나 프로그래밍을 같이 하는 것보다 더 효율적으로 텀 프로젝트를 할 수 있었다고 생각한다.

하드웨어의 경우, 처음에는 3D 프린터를 사용하려고 하였으나 소요 시간과 비용의 문제로 폼보드로 제작하게 되었다. 처음에는 기판을 세우는 방식으로 모델링하였으나, 폼보드가 기판의 무게를 제대로 지탱하지 못할 것이며, 선 정리가 어지러워질 것이라고 생각하여 지금과 같은 1층 2층의 구조를 띄게 되었다.

소프트웨어의 경우, 그냥 길게 늘여서 코딩하는 것이 아닌 클래스를 사용하여 객체 지향 프로그래밍으로 코드를 구성하려고 노력했다. .h,.cpp 파일로 나누어 보기 쉽고 사용하기 쉽게 만들었다. 헤더 파일에서 메서들과 변수들을 정의하고, cpp 파일에서 이에 대한 구현을 진행하였다. 이렇게 코드를 짜면 더 편리하게 관련 함수들을 사용할 수 있다. 밑에서 볼 수 있듯이 센서들의 값을 받아오는 과정에서 ADS_CONTROL 과 Gyro에 만들어져있는 auto_get_sensor_value 함수 이용하여 쉽게 접근할 수 있었다.

또, 값을 받아오기 위해 시리얼 통신을 할 때에도 사용자가 int, double, 문자열 등의 형태에 상관없이 통신을 할 수 있도록 만들었다.

이렇게 이전 고급프로그래밍에서 배웠던 객체 지향 원칙인 캡슐화를 로봇학실험3 프로젝트에 적용하고 나니 세세한 디테일에 신경 쓰지 않아도 되어서 자잘한 실수가 확연히 줄었고, 코드 가독성도 높아졌다.



원래는 7-세그먼트 대신 LCD를 사용하려 하였다. LCD로 온도, 습도, 불쾌지수 등의 편의성 정보를 제공하게 하려 하였으나, 사용하려고 준비한 4개의 LCD가 잘 작동하지 않았다. 혹시 몰라 다른 조와 조건을 동일하게 하여 테스트하였음에도 이상하게 우리 조의 LCD는 작동하지 않았다. 고장의 정확한 원인을 찾지 못하여 어쩔 수 없이 7-세그먼트를 사용한 것이 아쉽다.

칼만 필터를 어느정도 구현까지는 했다. 그래서 이를 이용하여 가속도 값과 각속도 값을 칼만 필터를 이용해 필터링 하고 이를 이용해 진동 주기, 진폭, 진동 방향 등등 수면 분석을 위한 다양한 정보들을 유도해 내고자 했다. 하지만 이를 우리 요구 조건에 맞게 적용을 해야 하는데 시간이 부족하여 작용까지 못한 부분이 많이 아쉬웠다.

인공 신경망과 이에 대한 학습을 C++로 직접 만들어본 경험이 있어서 이번 Atmega128에서도 이 인공신경망을 구현해서 적용하고자 했다. 그런데 인공 신경망을 만들기 위해서는 행렬 연산이 필요하고 이에 대한 행렬을 구현을 쉽게 하기 위해 동적할당이 필요한 경우가 많다. 그래서 이를 사용하려고 했으나 Atmega128에서 new나 malloc과 같은 동적 할당 연산자를 거의 지원하지 않았고, 이에 대해서 찾아보니 메모리 자원이 한정적이어서 동적 할당 자체도 매우 추천하지 않는 것 같았다. 행렬 연산에 필요한 Eigen Library도 성능상 문제 때문에 적용하는 것이 많이 제한적이었다. 이렇게 성능상 이슈 때문에 쓰이지 못한 기능도 많이 있어서 이번 프로젝트를 하면서 Atmega128의 제한된 리소스가 많이 아쉬웠다. 다음에 시간이 된다면 최적화를 잘해서 Atmega128의 제한된 성능을 모두 활용할 수 있는 프로젝트를 하고 싶다.

그리고, 시간이 된다면 실제로 센서들을 침대에 부착하여 가속도, 자이로 센서들의 값과 숙면 관계가 어떤지 분석하여 측정된 센서 값을 바탕으로 숙면의 정도를 알려주고 싶었으나 두 값 사이의 관계를 찾지 못하였고, 2개의 센서값 만으로는 AI 학습을 시키기 어려웠다. 추후 다양한 센서들을 활용하여 이 부분을 개선해보고 싶다.