모바일&스마트 시스템(D)

라즈베리파이를 이용한 코로나 병실 제어 장치

결과 보고서

2091012 박수민

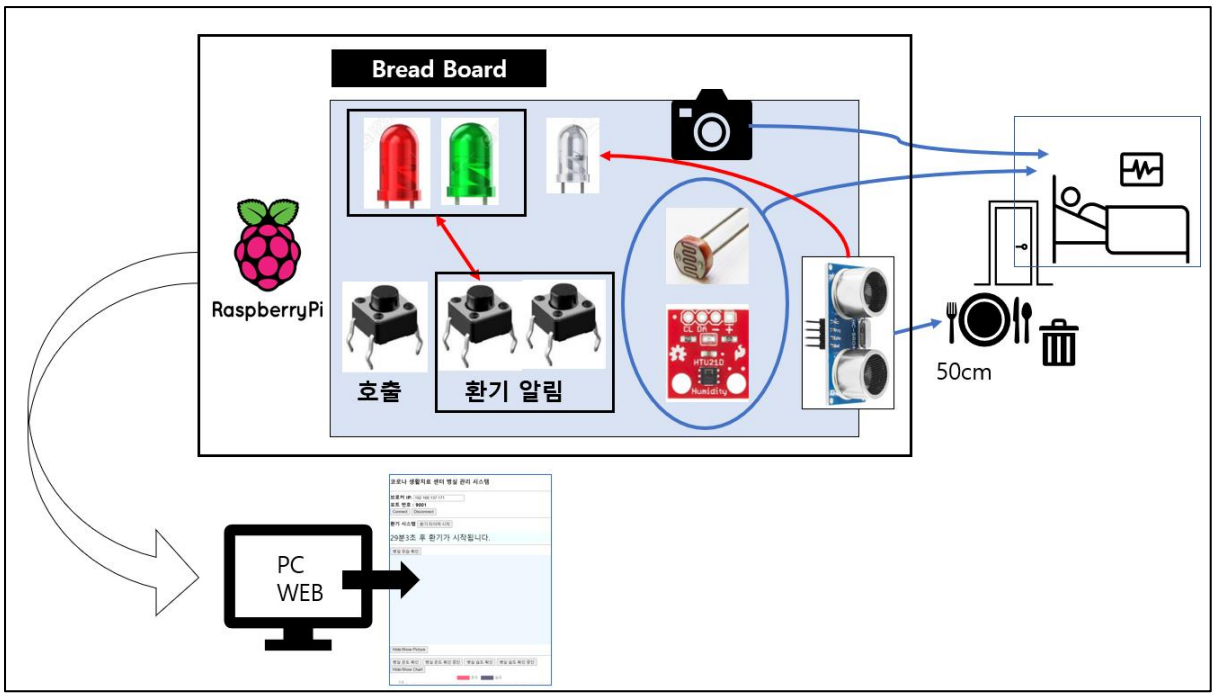
2021-12-04

목차

1. 작품 개요

이 장치는 '코로나 생활치료센터에 설치됨'이 전제된다. 라즈베리파이, LED, 스위치, 초음파 센서, 온습도 센서, 조도 센서, 카메라를 사용한다. 온습도 센서와 조도 센서를 사용하여 병실의 온도, 습도, 조도를 그래프로 컴퓨터에 출력하는 시스템을 사용한다. 카메라는 환자의 모습을 찍는데, 코로나라는 특수한 상황으로 인해 환자 병문안을 가지 못해 환자의 상태를 눈으로 확인하지 못하는 가족들을 위해 설치된 것이다. 초음파 센서는 병실 문에 부착되어 식사 배식을 하거나 쓰레기를 가지러 온 직원이 다가옴을 확인한다. 약 50cm 이내에서 초음파 센서가 인식하면 흰색 LED 가 켜진다. 스위치는 3 개를 사용한다. 병실 안에서 만약 응급 상황이 일어나면 첫 번째 스위치를 눌러 컴퓨터에 알람을 준다. 두 번째, 세 번째 스위치로 환기 타이머을 제어하는 시스템을 사용한다. 환기는 30 분 대기 -> 3 분 환기 -> 30 분 대기 -> 3 분 환기... 이렇게 진행된다. 먼저 [환기 타이머 시작] 버튼을 누르면 환기할 때까지 대기하는 30 분 타이머가 시작된다. 타이머가 끝나면 빨간색 LED 가 켜지는데, 이 때 두번째 스위치를 누르면 빨간색 LED 가 꺼지고 환기를 하는 3 분 타이머가 시작된다. 마찬가지로 3 분 타이머가 끝나면 초록색 LED 가 켜진다. 이 때 세번째 스위치를 누르면 초록색 LED 가 꺼지고 다시 30 분 타이머가 시작된다. 각 타이머의 남은 시간은 컴퓨터 웹페이지에서 확인 가능하다.

<최종 시스템 구성도>



2. 구현 방법

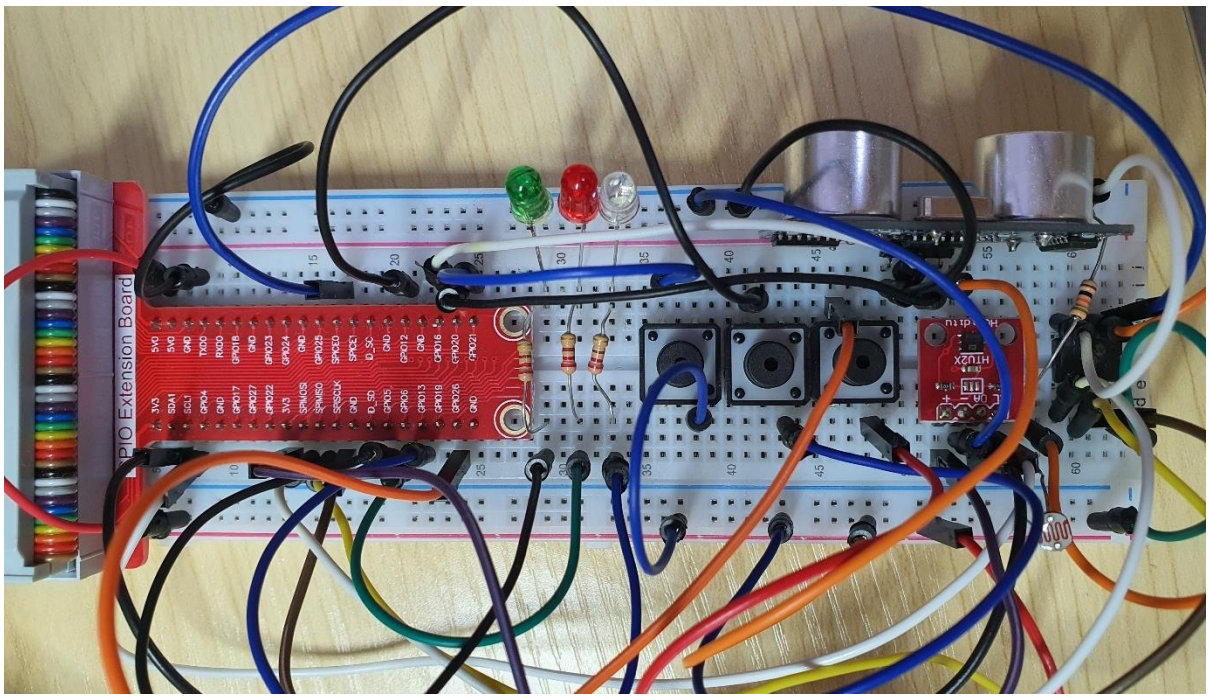
1) 하드웨어 부분

라즈베리 파이에 카메라 1 대, LED 3 개 스위치 3 개, 온습도센서/조도센서/초음파센서 각 1 대씩을 사용한다.

이들의 각 GPIO 핀은 다음과 같다.

- LED – GPIO 5, GPIO 6, GPIO 13
- 스위치 – GPIO 12, GPIO 21, GPIO 26
- 온습도 센서 – SDA1, SCL1
- 조도 센서 – SPIMOSI, SPIMISO, SPISCLK, SPICE0
- 초음파센서 – GPIO 16, GPIO 20

<실제 회로>



2) 소프트웨어 부분

2.2.1 proj_circuit.py

장치와 센서를 제어하는 함수가 내장된 파이썬 코드

```
# 필요한 모듈 import
import time
import RPi.GPIO as GPIO
from adafruit_htu21d import HTU21D
import busio
import Adafruit_MCP3008

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

# 초음파 센서 세팅
trig = 20
echo = 16
GPIO.setup(trig, GPIO.OUT)
GPIO.setup(echo, GPIO.IN)
GPIO.output(trig, False)

# 온습도 센서와 조도 센서 세팅
sda=2
scl=3
i2c=busio.I2C(scl, sda)
sensor=HTU21D(i2c)
mcp=Adafruit_MCP3008.MCP3008(clk=11, cs=8, miso=9, mosi=10)

# led GPIO 번호 세팅
ledG=5 #초록 led
ledR=6 #빨강 led
ledW=13 #흰색 led
GPIO.setup(ledG, GPIO.OUT) # 세 led 모두 출력 선으로 지정
GPIO.setup(ledR, GPIO.OUT)
GPIO.setup(ledW, GPIO.OUT)

button1=21 #버튼 1 GPIO 21 번
button2=12 #버튼 2 GPIO 12 번
button3=26 #버튼 3 GPIO 26 번
GPIO.setup(button1, GPIO.IN, GPIO.PUD_DOWN) # 세 버튼 모두 풀다운 효과 지장
GPIO.setup(button2, GPIO.IN, GPIO.PUD_DOWN)
GPIO.setup(button3, GPIO.IN, GPIO.PUD_DOWN)

# 초음파센서를 사용해 거리 측정
def measureDistance():
```

```
        global trig, echo
        GPIO.output(trig, True) # 신호 1 발생
        time.sleep(0.00001) # 짧은시간후 0으로 떨어뜨려 falling edge를 만
        GPIO.output(trig, False) # 신호 0 발생(falling 에지)

        while(GPIO.input(echo) == 0):
                pass
        pulse_start = time.time() # 신호 1. 초음파 발생이 시작되었음을 알림
        while(GPIO.input(echo) == 1):
                pass
        pulse_end = time.time() # 신호 0. 초음파 수신 완료를 알림

        pulse_duration = pulse_end - pulse_start
        return 340*100/2*pulse_duration

# 온도 리턴
def getTemperature():
    return float(sensor.temperature)
# 습도 리턴
def getHumidity():
    return float(sensor.relative_humidity)
#led를 키고 끄는 함수
def ledOnOff(led, onOff):
    GPIO.output(led, onOff)
#빨강 led on 함수
def ledROnOff():
    GPIO.output(ledR, 1)
#초록 led on 함수
def ledGOnOff():
    GPIO.output(ledG, 1)
#빨강 led off 함수
def ledROff():
    GPIO.output(ledR, 0)
#초록 led off 함수
def ledGOff():
    GPIO.output(ledG, 0)

#각 버튼의 누름을 인식하는 함수
def button1Pressed(button1):
    return True

def button2Pressed(button2):
    return True

def button3Pressed(button3):
    return True
```

2.2.2 proj_mqtt.py

mqtt 를 사용해 데이터를 전송하는 파이썬 코드

```
import time
import paho.mqtt.client as mqtt
import myCamera
import proj_circuit

# 버튼과 흰색 led 초기 설정
flag = False
btn1 = False
btn2 = False
btn3 = False
ledWStatus = 0

# on_connect 콜백 함수
def on_connect(client, userdata, flag, rc):
        # 얼굴인식(카메라), ledR, ledG 를 subscribe
        client.subscribe("facerecognition", qos=0)
        client.subscribe("ledR", qos=0)
        client.subscribe("ledG", qos=0)

# on_message 콜백 함수
def on_message(client, userdata, msg) :

        global flag
        command=msg.payload.decode("utf-8")
        # 메시지가 action 이면 다음 문구 출력 후 flag 에 True 준다.
        if command=="action":
            print("action msg received..")
            flag=True
        # ledR 면 빨강색 led 킨다.
        elif command=="ledR":
            proj_circuit.ledROnOff()
        # ledG 면 초록색 led 킨다.
        elif command=="ledG":
            proj_circuit.ledGOnOff()

broker_ip = "localhost" # 현재 이 컴퓨터를 브로커로 설정

client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message

client.connect(broker_ip, 1883)
```

```
client.loop_start()

while True :
        # 온도, 습도, 조도 토픽에 각 함수를 사용해 데이터를 publish 한다.
        client.publish("temp", proj_circuit.getTemperature(), qos=0)
        client.publish("humid", proj_circuit.getHumidity(), qos=0)
        client.publish("illumi",proj_circuit.mcp.read_adc(0), qos=0)

        # 위 명령이 action 이여서 flag 가 True 가 되면
        #myCamera 의 함수를 사용해 사진을 찍어
        # 토픽 image 로 사진을 publish 한다. publish 한 후 flag 를 False 로 한다.
        if flag==True:
            imageFileName = myCamera.takePicture()
            client.publish("image", imageFileName, qos=0)
            flag=False

        # 첫 번째 버튼이 눌리면 alert 토픽에 메시지 True 을 publish 한다.
        btn1 = proj_circuit.GPIO.input(proj_circuit.button1)
        if btn1==1:
            client.publish("alert", btn1, qos=0)
            btn1=False

        # 두 번째 버튼이 눌리면 timer30 토픽에 메시지 True 을 publish 한다.
       # 30 분 타이머가 끝났다는 것을 의미
        btn2 = proj_circuit.GPIO.input(proj_circuit.button2)
        if btn2==1:
            proj_circuit.ledROff()
            client.publish("timer30", btn2, qos=0)
            btn2=False

        # 세 번째 버튼이 눌리면 timer5 토픽에 메시지 True 을 publish 한다.
       # 3 분 타이머가 끝났다는 것을 의미
        btn3 = proj_circuit.GPIO.input(proj_circuit.button3)
        if btn3==1:
            proj_circuit.ledGOff()
            client.publish("timer5", btn3, qos=0)
            btn3=False

        #초음파 센서를 사용해 거리를 잰다.
        #50cm 를 넘으면 흰색 led 를 끄고 넘으면 킨다.
        distance = proj_circuit.measureDistance()
        if(distance>50):
            onOff=0
            proj_circuit.ledOnOff(proj_circuit.ledW, onOff)

        else:
            onOff=1
            proj_circuit.ledOnOff(proj_circuit.ledW, onOff)
```

```
        time.sleep(1)


client.loop_stop()
client.disconnect()
```

2.2.3 proj_app.py

웹 브라우저로부터 접속과 요청을 받아 처리하는 플라스크 앱

```
from flask import Flask, render_template, request
app=Flask(__name__)

@app.route('/')
def index():
    # '/'로 접속하면 project.html를 랜더링한다.
    return render_template('project.html')

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=8080)
```

2.2.4 myCamera.py

```
import os
import io
import time
import picamera
import cv2
import numpy as np

# 전역 변수 선언 및 초기화
fileName = ""
stream = io.BytesIO()
haar = cv2.CascadeClassifier('./haarCascades/haar-cascade-\
files-master/haarcascade_frontalface_default.xml')

camera = picamera.PiCamera()
camera.resolution = (550, 350)
time.sleep(1) # 카메라 워밍업
```

```
# 카메라로 사진 촬영 후 opencv 를 이용하여 얼굴을 인식하여 얼굴에
#사각형으로 그리고 jpg 파일로 저장, jpg 파일 이름 리턴
def takePicture() :
        global fileName
        global stream
        global camera

        # 이전에 만들어둔 사진 파일이 있으면 삭제
        if len(fileName) != 0:
                os.unlink(fileName)

       # 파일 포인터를 스트림 맨 앞으로 위치시킴. 이곳에서부터 이미지 데이터 저장
        stream.seek(0)
        stream.truncate()
        camera.start_preview() # 연결된 모니터에 보이도록 하기 위함
        time.sleep(3) # 사용자가 카메라  앞에서 얼굴 들이미는 동안 시간 주기
        camera.capture(stream, format='jpeg', use_video_port=True)
        camera.stop_preview()
        data = np.frombuffer(stream.getvalue(), dtype=np.uint8)
        image = cv2.imdecode(data, 1)
        image_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        faces = haar.detectMultiScale(image_gray,1.1,3)

        for x, y, w, h in faces:
                cv2.rectangle(image, (x, y), (x + w, y + h), (255, 0, 0), 2)

        takeTime = time.time()
        fileName = "./static/%d.jpg" % (takeTime * 10)
        cv2.imwrite(fileName, image)
        return fileName

if __name__ == '__main__' :
        takePicture()

if __name__ == '12-3camera' :
        pass
```

2.2.5 mqttio5.js

MQTT 브로커에 접속하고 메시지를 송수신할 수 있는 js 코드

```
var port = 9001 // mosquitto 의 디폴트 웹 포트
var client = null; // null 이면 연결되지 않았음

function startConnect() { // 접속을 시도하는 함수
    clientID = "clientID-" + parseInt(Math.random() * 100); // 랜덤한 사용자 ID
생성
    // 사용자가 입력한 브로커의 IP 주소와 포트 번호 알아내기
    broker = document.getElementById("broker").value; // 브로커의 IP 주소
    // id 가 message 인 DIV 객체에 브로커의 IP 와 포트 번호 출력
    // MQTT 메시지 전송 기능을 모두 가징 Paho client 객체 생성
    client = new Paho.MQTT.Client(broker, Number(port), clientID);
    // client 객체에 콜백 함수 등록
    client.onConnectionLost = onConnectionLost; // 접속이 끊어졌을 때 실행되는
함수 등록
    client.onMessageArrived = onMessageArrived; // 메시지가 도착하였을 때
실행되는 함수 등록
    // 브로커에 접속. 매개변수는 객체 {onSuccess : onConnect}로서, 객체의
프로퍼틴느 onSuccess 이고 그 값이 onConnect.
    // 접속에 성공하면 onConnect 함수를 실행하라는 지시
    client.connect({
    onSuccess: onConnect,
    });
}

var isConnected = false;

function onConnect() { // 브로커로의 접속이 성공할 때 호출되는 함수
    isConnected = true;
    document.getElementById("messages").innerHTML +=
'<span>Connected</span><br/>';
}

var topicSave;
function subscribe(topic) {
    if(client == null) return;
    if(isConnected != true) {
        topicSave = topic;
        window.setTimeout("subscribe(topicSave)", 500);
        return
    }
    // 토픽으로 subscribe 하고 있음을 id 가 message 인 DIV 에 출력
    //document.getElementById("messages").innerHTML += '<span>Subscribing to:
' + topic + '</span><br/>';
```

```
    client.subscribe(topic); // 브로커에 subscribe
}

function publish(topic, msg) {
    if(client == null) return; // 연결되지 않았음
    client.send(topic, msg, 0, false);
}

function unsubscribe(topic) {
    if(client == null || isConnected != true) return;
    // 토픽으로 subscribe 하고 있음을 id 가 message 인 DIV 에 출력
    //document.getElementById("messages").innerHTML += '<span>Unsubscribing to:
' + topic + '</span><br/>';
    client.unsubscribe(topic, null); // 브로커에 subscribe
}

// 접속이 끊어졌을 때 호출되는 함수
function onConnectionLost(responseObject) { // 매개변수인 responseObject 는 응답
패킷의 정보를 담은 개체
    document.getElementById("messages").innerHTML += '<span>오류 : 접속
끊어짐</span><br/>';
    if (responseObject.errorCode !== 0) {
    document.getElementById("messages").innerHTML += '<span>오류 : ' + +
responseObject.errorMessage + '</span><br/>';
    }
}

// 환기 30 분 대기 타이머, timer5 토픽을 구독한다.
function startTimer1() {
    subscribe("timer5");
    var time1 = 1800; //30 분 시간 설정
    var min="";
    var sec="";

    //setInterval 함수 사용
    var x=setInterval(function() {
        min=parseInt(time1/60);
        sec=time1%60;

        document.getElementById("timer").innerHTML=min+"분"+sec+"초 후 환기가
시작됩니다.";
        time1--;

        // 타이머가 끝나면 "ledR"를 구독하고 "ledR"를 보낸다
        // 그로 인해 빨강색 led 가 켜진다.
        if(time1<0) {
            clearInterval(x);
            document.getElementById("timer").innerHTML="환기 시작";
```

```
            publish("ledR" ,"ledR");
        }
    }, 2000);
}

//환기 시작 3분 타이머
function startTimer2() {
    var time2 = 180; //3분 시간 설정
    var min="";
    var sec="";

    var x=setInterval(function() {
        min=parseInt(time2/60);
        sec=time2%60;

        document.getElementById("timer").innerHTML=min+"분"+sec+"초 후 환기가
종료됩니다.";
        time2--;

        //타이머가 끝나면 "ledG"를 구독하고 "ledG"를 보낸다
        //그로 인해 초록색 led가 켜진다
        if(time2<0) {
            clearInterval(x);
            document.getElementById("timer").innerHTML="환기 종료";
            publish("ledG" ,"ledG");
        }
    }, 2000);
}

// 메시지가 도착할 때 호출되는 함수
function onMessageArrived(msg) { // 매개변수 msg는 도착한 MQTT 메시지를 담고 있는
객체
    console.log("onMessageArrived: " + msg.payloadString);

    // 토픽 image가 도착하면 payload에 담긴 파일 이름의 이미지 그리기
    if(msg.destinationName == "image") {
    drawImage(msg.payloadString); // 메시지에 담긴 파일 이름으로 drawImage() 호출.
drawImage()는 웹 페이지에 있음
    }
    if(msg.destinationName=="illumi") {
        addChartDataIllumi(parseFloat(msg.payloadString)); // 받은 조도 값을
조도 그래프를 그리는 함수로 보낸다
    }
    if(msg.destinationName=="temp") {
        // 받은 온도 값을 온습도 그래프를 그리는 함수로 보낸다.
        // 온습도 그래프의 0번
        addChartDataTemp(0,parseFloat(msg.payloadString));
    }
```

```
    if(msg.destinationName=="humid") {
        //받은 습도 값을 온습도 그래프를 그리는 함수로 보낸다.
        // 온습도 그래프의 1 번
        addChartDataTemp(1, parseFloat(msg.payloadString));
    }
    if(msg.destinationName=="alert") {
        //alert 토픽이 도착하면 alert 함수로 웹페이지에 알림을 보낸다.
    window.alert("환자 호출, 확인바랍니다.")
    }
    if(msg.destinationName=="timer30") {
        //timer30 토픽은 30 분 타이머가 끝났다는 것을 의미하므로
        //timer30 토픽이 도착하면 3 분 타이머를 시작시킨다.
        startTimer2();
    }
    if(msg.destinationName=="timer5") {
        //timer5 토픽은 3 분 타이머가 끝났다는 것을 의미하므로
        // timer5 토픽이 도착하면 30 분 타이머를 시작시킨다.
        startTimer1();
    }

}

// disconnection 버튼이 선택되었을 때 호출되는 함수
function startDisconnect() {
    client.disconnect(); // 브로커에 접속 해제
    document.getElementById("messages").innerHTML +=
'<span>Disconnected</span><br/>';
}
```

2.2.6 face.js

```
// 전역 변수 선언
var canvas;
var context;
var img;

// load 이벤트 리스너 등록. 웹페이지가 로딩된 후 실행
window.addEventListener("load", function() {
    canvas = document.getElementById("canvasCam");
    context = canvas.getContext("2d");

    img = new Image();
    img.onload = function () {
        context.drawImage(img, 0, 20); // (0,0) 위치에 img 의 크기로 그리기
```

```
    }
});

// drawImage()는 "image' 토픽이 도착하였을 때 onMessageArrived()에 의해 호출된다.
function drawImage(imgUrl) { // imgUrl 은 이미지의 url
    img.src = imgUrl; // img.onload 에 등록된 코드에 의해 그려짐
}

var isImageSubscribed = false;
function recognize() {
        if(!isImageSubscribed) {
                subscribe('image'); // 토픽 image 등록
                isImageSubscribed = true;
        }
        publish('facerecognition', 'action'); // 토픽: facerecognition, action
메시지 전송
}
```

2.2.7 proj_myChart.js

각 그래프들을 그릴 수 있는 함수가 내장된 js 코드

```
// Chart 객체에 넘겨줄 차트에 대한 정보들을 정의한 객체.

//온습도 그래프 설정
var configTemp = {
        // type 은 차트 종류 지정
        type: 'line',

        // data 는 차트에 출력될 전체 데이터 표현
        data: {
                // labels 는 배열로 데이터의 레이블들
                labels: [],

                // datasets 배열로 이 차트에 그려질 모든 데이터 셋 표현.
                //datasets[0] -> 온도 그래프
                //datasets[1] -> 습도 그래프
                datasets: [{
                        label: '온도',
                        backgroundColor: 'yellow',
                        borderColor: 'rgb(255, 99, 132)',
                        borderWidth: 2,
                        data: [], /* 각 레이블에 해당하는 데이터 */
                        fill : false,
```

```
                },
                {
                    label: '습도',
                    backgroundColor: 'blue',
                    borderColor: 'rgb(100, 99, 132)',
                    borderWidth: 2,
                    data: [], /* 각 레이블에 해당하는 데이터 */
                    fill : false,
            }]
        },

        //  차트의 속성 지정
        options: {
                responsive : false, // 크기 조절 금지
                scales: { /* x 축과 y 축 정보 */
                        xAxes: [{
                                display: true,
                                scaleLabel: { display: true, labelString:
'시간' },
                        }],
                        yAxes: [{
                                display: true,
                                scaleLabel: { display: true, labelString:
'수치' }
                        }]
                }
        }
    };

    //조도 그래프 설정
    var configIllumi = {
        // type 은 차트 종류 지정
        type: 'line', /* line 등으로 바꿀 수 있음 */

        // data 는 차트에 출력될 전체 데이터 표현
        data: {
                // labels 는 배열로 데이터의 레이블들
                labels: [],

                // datasets 배열로 이 차트에 그려질 모든 데이터 셋 표현. 아래는
그래프 1 개만 있는 경우
                datasets: [{
                        label: '조도',
                        backgroundColor: 'yellow',
                        borderColor: 'rgb(255, 99, 132)',
                        borderWidth: 2,
                        data: [], /* 각 레이블에 해당하는 데이터 */
```

```
                        fill : false, /* 그래프 아래가 채워진 상태로 그려집니다.
해보세요 */
                }]
        },

        //  차트의 속성 지정
        options: {
                responsive : false, // 크기 조절 금지
                scales: { /* x 축과 y 축 정보 */
                        xAxes: [{
                                display: true,
                                scaleLabel: { display: true, labelString:
'시간' },
                        }],
                        yAxes: [{
                                display: true,
                                scaleLabel: { display: true, labelString:
'수치' }
                        }]
                }
        }
    };

    var ctxTemp = null
    var ctxIllumi = null

    var chartTemp = null
    var chartIllumi = null

    var LABEL_SIZE = 20; // 차트에 그려지는 데이터의 개수
    var tick1 = 0; // 도착한 데이터의 개수임, tick 의 범위는 0 에서 99 까지만
    var tick2 = 0;
    var label;

    // 캔버스를 가져와서 그래프를 그릴 준비를 한다
    function drawChartTemp() {
        ctxTemp = document.getElementById('canvasTemp').getContext('2d');
        chartTemp = new Chart(ctxTemp, configTemp);
        initTemp();
    } // end of drawChart()

    // 캔버스를 가져와서 그래프를 그릴 준비를 한다
    function drawChartIllumi() {
        ctxIllumi = document.getElementById('canvasillumi').getContext('2d');
        chartIllumi = new Chart(ctxIllumi, configIllumi);
        initIllumi();
    } // end of drawChart()
```

```
    // chart의 차트에 labels의 크기를 LABEL_SIZE로 만들고 0~19까지 레이블 붙이기
    function initTemp() {
        for(let i=0; i<LABEL_SIZE; i++) {
                chartTemp.data.labels[i] = i;
        }
        chartTemp.update();
    }

    function initIllumi() {
        for(let i=0; i<LABEL_SIZE; i++) {
                chartIllumi.data.labels[i] = i;
        }
        chartIllumi.update();
    }

    // 조도 그래프 그림
    function addChartDataIllumi(value) {
        tick1++; // 도착한 데이터의 개수 증가
        tick1 %= 100; // tick의 범위는 0에서 99까지만. 100보다 크면 다시 0부터
시작
        let n = chartIllumi.data.datasets[0].data.length; // 현재 데이터의 개수
        if(n < LABEL_SIZE)
                chartIllumi.data.datasets[0].data.push(value);
        else {
                // 새 데이터 value 삽입
                chartIllumi.data.datasets[0].data.push(value);
                chartIllumi.data.datasets[0].data.shift();

                // 레이블 삽입
                chartIllumi.data.labels.push(tick1);
                chartIllumi.data.labels.shift();
        }
        chartIllumi.update();
    }

    //label이 0이면 온도 그래프 그림
    //label이 1이면 습도 그래프 그림
    function addChartDataTemp(label, value) {
        tick2++; // 도착한 데이터의 개수 증가
        tick2 %= 100; // tick의 범위는 0에서 99까지만. 100보다 크면 다시 0부터
시작
        let n1 = chartTemp.data.datasets[label].data.length; // 현재 데이터의
개수
        if(n1 < LABEL_SIZE) {
                chartTemp.data.datasets[label].data.push(value);
        }
        else {
                // 새 데이터 value 삽입
```

```
                chartTemp.data.datasets[label].data.push(value);
                chartTemp.data.datasets[label].data.shift();

                // 레이블 삽입
                chartTemp.data.labels.push(tick2);
                chartTemp.data.labels.shift();
            }
            chartTemp.update();
        }


        function hideshowCam() { // 카메라 캔버스 보이기 숨기기
            if(canvasCam.style.display == "none")  canvasCam.style.display =
"block"
            else canvasCam.style.display = "none"
        }
        function hideshowTemp() { // 온습도 그래프 캔버스 보이기 숨기기
            if(canvasTemp.style.display == "none")  canvasTemp.style.display =
"block"
            else canvasTemp.style.display = "none"
        }
        function hideshowIllumi() { // 조도 그래프 캔버스 보이기 숨기기
            if(canvasillumi.style.display == "none")  canvasillumi.style.display =
"block"
            else canvasillumi.style.display = "none"
        }

        window.addEventListener("load", drawChartTemp); // load 이벤트가 발생하면
drawChartTemp() 호출하도록 등록
        window.addEventListener("load", drawChartIllumi); // load 이벤트가 발생하면
drawChartIllumi() 호출하도록 등록
```

2.2.8 myStyle.css

타이머가 작동될 때의 배경색 설정

```
#timer {
    font-size: 1.5em;
    background-color: azure;
}
```

2.2.9 project.html

센서의 값을 읽고 모니터 할 수 있는 <코로나 생활 치료 센터 병실 관리> 웹페이지

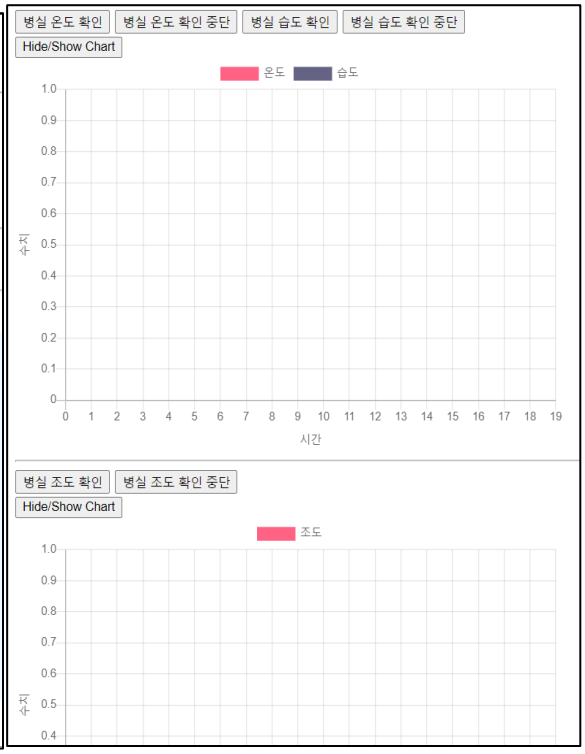
```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>코로나 생활치료 센터 병실 관리</title>
<meta charset="utf-8">
<script src="https://cdnjs.cloudflare.com/ajax/libs/paho-
mqtt/1.0.2/mqttws31.min.js" type="text/javascript"></script>
<script src="./static/mqttio5.js" type="text/javascript"></script>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.9.4/Chart.min.js"
type="text/javascript"></script>
<script src="./static/proj_myChart.js" type="text/javascript"></script>
<script src="./static/face.js" type="text/javascript"></script>
<link type="text/css" rel="stylesheet" href="./static/myStyle.css">
<script>
window.addEventListener("load", function () {
// http://224..129:8080/에서 224...의 IP 만 끊어내는 코드
var url = new String(document.location);
ip = (url.split("//"))[1]; // ip = "224...:8080/"
ip = (ip.split(":"))[0]; // ip = "224..."
document.getElementById("broker").value = ip
});


</script>
</head>

<body>
<h3>코로나 생활치료 센터 병실 관리 시스템</h3>
<hr>
<form id="connection-form">
<b>브로커 IP:</b>
<input id="broker" type="text" name="broker" value=""><br>
<b>포트 번호 : 9001</b><br>
<!-- Connect 버튼을 누르면 접속을 시도하고 alert 를 구독하기 시작한다. -->
<input type="button"
onclick="startConnect(),subscribe('alert')"  value="Connect">
<!-- Disconnect 버튼을 누르면 접속이 끊긴다. -->
<input type="button" onclick="startDisconnect()" value="Disconnect">
</form>
<div id="messages"></div>
<hr>
```

```
<form id="LED-control-form">
<b>환기 시스템  </b>
<!-- 버튼을 누르면 30 분 타이머를 시작하고 토픽 timer30 을 구독하기 시작한다. -->
<input type="button" onclick="startTimer1(), subscribe('timer30')" value="환기
타이머 시작">
<p></p>
<!-- 이곳에 남은 타이머 시간과 환기 종료/ 환기 시작 문구가 뜬다-->
<div id="timer"></div>
</form>
<hr>
<!-- 카메라 폼 -->
<form id="camera-form">
<!-- 병실 모습 확인 버튼으로 사진을 캔버스에 올린다.-->
<input type="button" onclick="recognize()" value="병실 모습 확인">
</form>
<canvas id="canvasCam" style="background-color: aliceblue" width="550"
height="300"></canvas><br>
<!-- 버튼을 누르면 캔버스가 숨겨진다-->
<button id="hideshow" onclick="hideshowCam()">Hide/Show Picture</button>
<div id="messagesCam"></div>
<hr>
<!-- 각 버튼으로 온습도 토픽을 구독하고 구독 해지한다-->
<form id="temp-form">
    <input type="button" onclick="subscribe('temp')", value="병실 온도 확인">
    <input type="button" onclick="unsubscribe('temp')" value="병실 온도 확인
중단">
    <input type="button" onclick="subscribe('humid')", value="병실 습도 확인">
    <input type="button" onclick="unsubscribe('humid')" value="병실 습도 확인
중단">
</form>
<!-- 버튼을 누르면 캔버스가 숨겨진다-->
<button id="hideshow" onclick="hideshowTemp()">Hide/Show Chart</button>
<canvas id="canvasTemp" width="600" height="400"></canvas>
<hr>
<!-- 각 버튼으로 조도 토픽을 구독하고 구독 해지한다. -->
<form id="illumi-form">
    <input type="button" onclick="subscribe('illumi')" value="병실 조도 확인">
    <input type="button" onclick="unsubscribe('illumi')" value="병실 조도 확인
중단">
</form>
<!-- 버튼을 누르면 캔버스가 숨겨진다-->
<button id="hideshow" onclick="hideshowIllumi()">Hide/Show Chart</button>
<canvas id="canvasillumi" width="600" height="400"></canvas>
</body>
</html>
```

3. 실행 과정 및 결과

3.1 초기

192.168.137.171:8080 에 접속하면 아래 사진과 같이 웹페이지가 켜지고 [Connect] 버튼을 누르면 연결됐다는 문구가 뜬다.



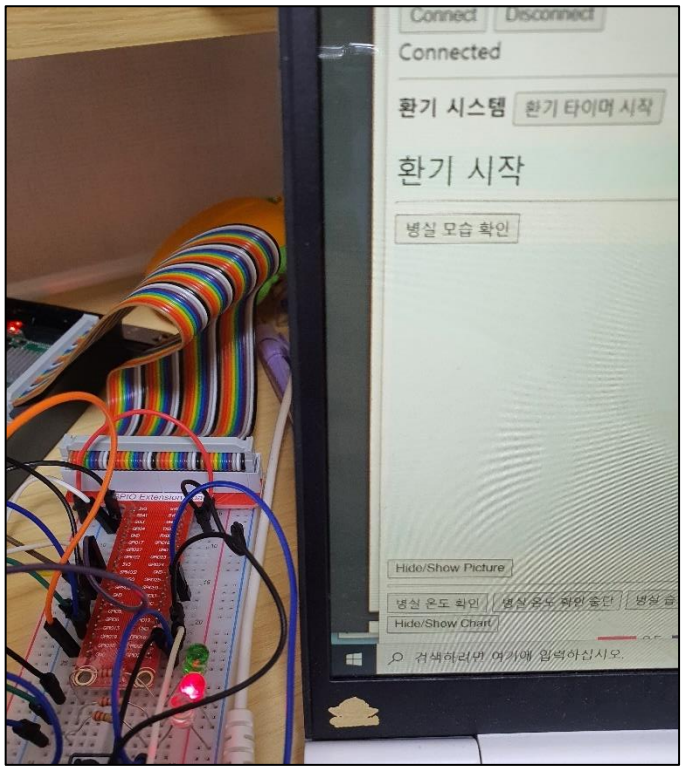
3.2 환기 시스템

3.2.1 [환기 타이머 시작] 버튼을 누른 경우

다음과 같은 문구가 뜨며 30 분 타이머가 시작된다.

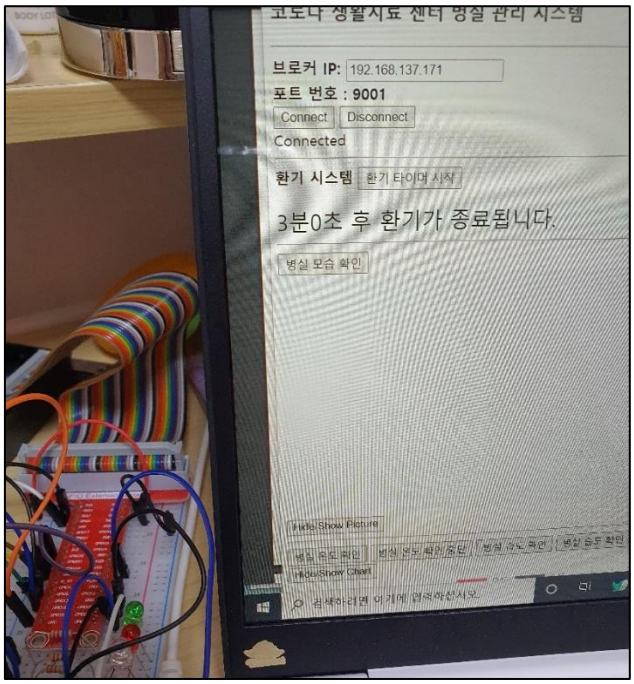
3.2.2 30 분 타이머가 끝난 후

<환기 시작> 문구와 함께 빨간색 LED 가 켜진다.



3.2.3 두 번째 스위치를 누른 경우

빨간색 LED 가 꺼지고 아래 사진과 같이 환기를 하는 3 분 타이머가 시작된다.

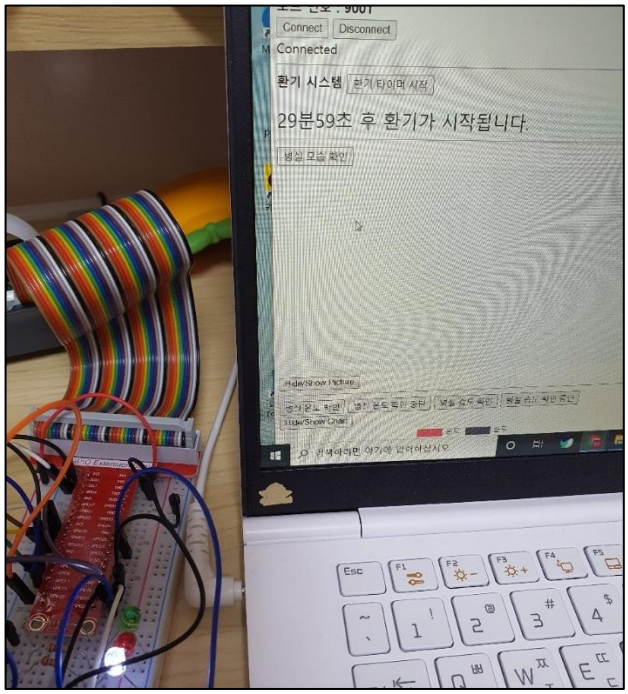


3.2.4 3 분 타이머가 끝난 후

<환기 종료> 문구와 함께 초록색 LED 가 켜진다.

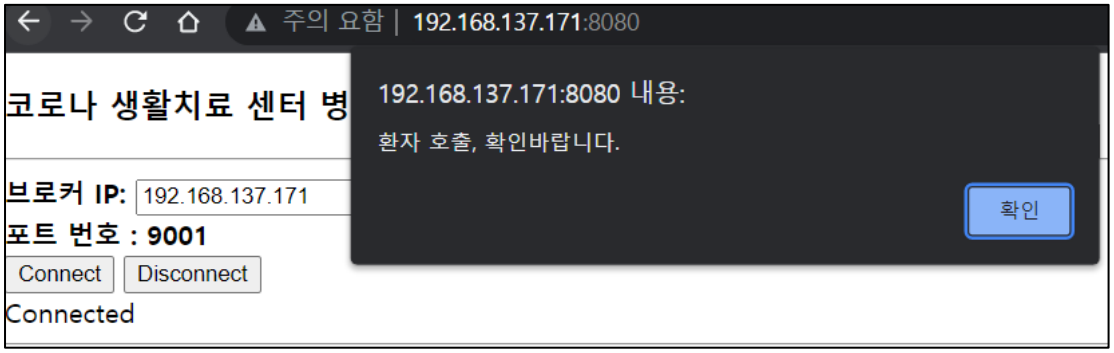
3.2.5 세 번째 스위치를 누른 경우

초록색 LED 가 꺼지고 환기할 때까지 대기하는 30 분 타이머가 다시 시작된다.



3.3 첫 번째 스위치를 누른 경우

다음 사진과 같은 문구와 함께 웹페이지에 알람이 발생한다.



3.4 병실 모습 확인

3.4.1 [병실 모습 확인] 버튼을 눌렀을 때

병실과 환자의 모습이 찍힌다.

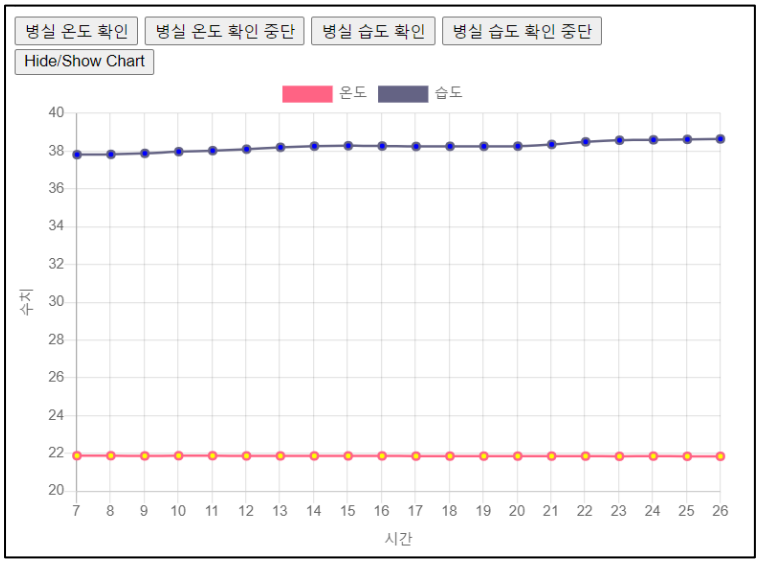
3.4.2 [Hide/Show Picture] 버튼을 눌렀을 때

사진을 담고 있는 캔버스가 숨겨진다. 버튼을 한 번 더 누르면 다시 나타난다.

3.5 온/습도 확인

3.5.1 [병실 온도 확인], [병실 습도 확인] 버튼을 눌렀을 때

누른 버튼에 따라 온도나 습도 그래프가 그려지며 둘 다 누르면 아래 사진과 같이 그래프가 그려진다.

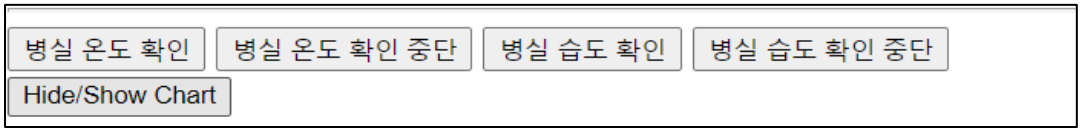


3.5.2 [병실 온도 확인 중단], [병실 습도 확인 중단] 버튼을 눌렀을 때

누른 버튼에 해당하는 그래프가 멈추고 더 이상 그래프를 그리지 않는다.

3.5.3 [Hide/Show Chart] 버튼을 눌렀을 때

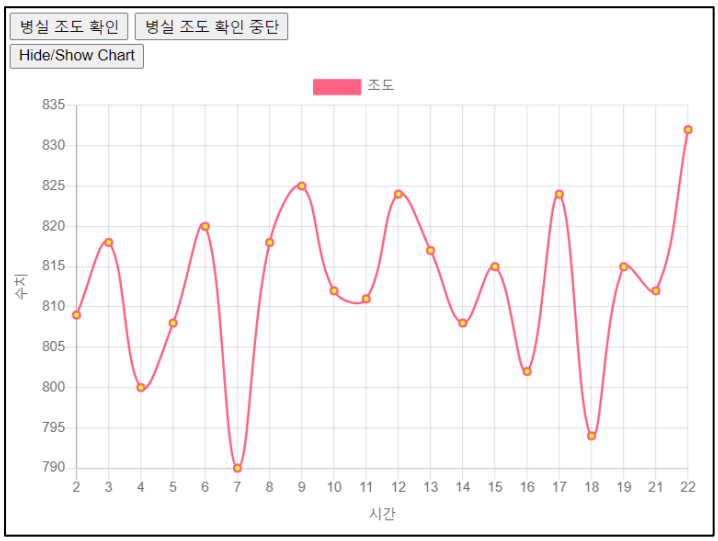
그래프가 숨겨진다. 버튼을 한 번 더 누르면 다시 그래프가 보인다.



3.6 조도 확인

3.6.1 [병실 조도 확인] 버튼을 눌렀을 때

병실의 조도 수치를 보여주는 그래프가 그려진다.



3.6.2 [병실 조도 확인 중단] 버튼을 눌렀을 때

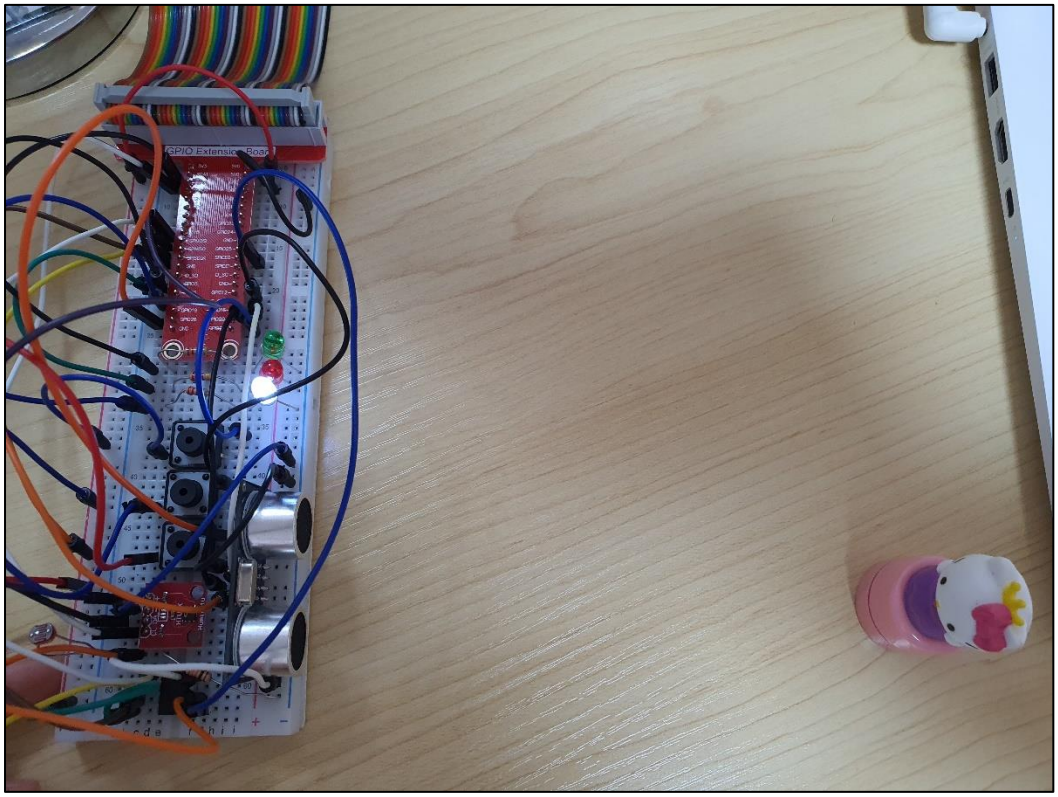
조도 그래프가 멈추고 더 이상 그래프를 그리지 않는다.

3.6.3 [Hide/Show Chart] 버튼을 눌렀을 때

그래프가 숨겨진다. 버튼을 한 번 더 누르면 다시 그래프가 보인다.

3.7 초음파 센서 50cm 이내에 사람이 들어왔을 때

흰색 LED 에 불이 들어온다.



3.8 연결 해제

[Disconnect] 버튼을 누르면 접속이 끊긴다.

4. 결론

모바일&스마트 시스템 과목을 배우면서 센서들을 제어하고 데이터를 다룰 수 있게 되었으며 mqtt 와 flask 앱을 가지고 웹 페이지와 통신하는 법을 알게 되었다. 제안서 -> 구현 -> 보고서 과정을 겪으면서 컴퓨터 공학과 학생으로서 하나의 잘 정돈된 프로젝트를 완성하는 방법을 알게 되었다. 프로젝트를 구현하는 과정 중 mqttio5.js 와 proj_circuit.py 의 버튼을 다루는 부분에서 어려움이 있었지만, mqtt 의 알고리즘을 정확히 이해하려고 공부하다 보니 문제들을 자연스럽게 해결할 수 있었다.

이 과목에서 주로 사용했던 파이썬에 대한 공부가 조금 부족하다는 것을 인지해 파이썬 언어에 대한 공부의 필요성을 느꼈다. 미래에 훌륭한 컴퓨터 전문가가 되기 위해서는 모르는 부분은 정확히 알 때까지 파고들어 이해해야 함을 깨닫게 되었다.