

# **모바일 & 스마트시스템[B]**

## **< 미니프로젝트 보고서 >**

**빅데이터트랙**

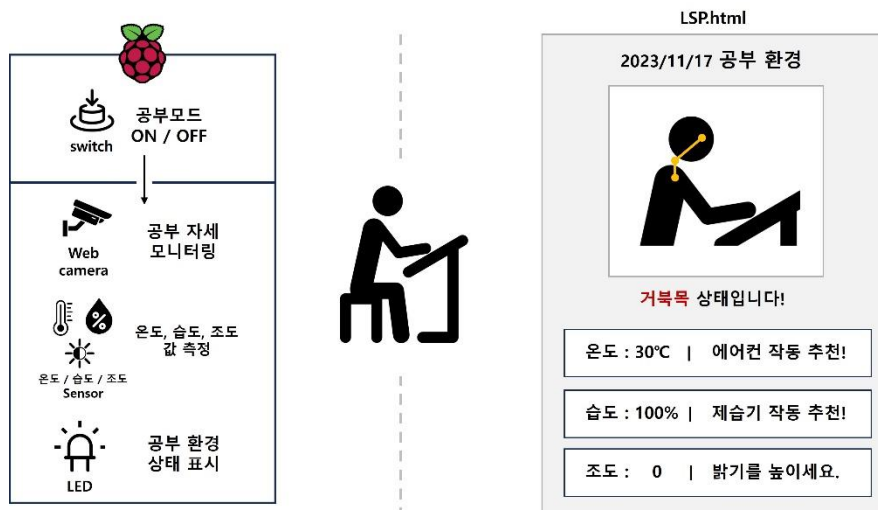
**1871027 김동준**

# 라즈베리파이를 이용한 “올바른 공부 환경 조성 시스템”

## 1. 작품 개요

이 프로젝트는 라즈베리파이를 중심으로 구축된 '공부 도우미' 시스템으로, 학생 및 직장인들이 효율적으로 공부하고 작업할 수 있는 환경을 조성하는 것을 목표로 한다. 라즈베리파이의 브레드 보드에 연결된 LED센서, 스위치, 온도, 습도, 조도 센서와 USB로 연결된 웹 카메라를 사용하며, Flask 웹 서버를 통해 구현한 Webpage를 사용자에게 서비스한다. 스위치를 누를 시 “공부 모드”가 시작되며 스위치를 다시 누를 시 공부 모드가 종료된다. 공부 모드에선 LED1이 켜지고, mosquitto broker을 이용한 mqtt통신을 통해 서버에게 공부 장소의 온도, 습도, 조도를 송신하고 웹 카메라에 찍힌 사용자의 모습을 송신한다. 송신 간격은 서버의 연산 속도를 고려하여, 특정 간격을 두고 송신한다. 서버는 라즈베리파이의 웹 카메라를 통해 송신 받은 이미지를 OpenPose(Pose Estimation) AI 모델을 이용하여 사용자의 관절 포인트를 인식하게 되고, 관절간 각도 계산 코드를 통해 거북목 상태 유무를 판단하게 된다. 이에 따라, 거북목 상태 발생시 웹페이지에 표시하게 된다. 또한, 사용자가 설정한 온도, 습도, 조도의 임계 값에 벗어나는 환경이 발생시, LED2에 표시하게 된다. Flask 웹서버를 통해 구현한 Webpage에선 현재 공부 환경 상태를 나타내며, 정상 범위에서 벗어난 상황인 경우(거북목 상태, 온도 / 습도 / 조도 비정상 ) 이를 표시하게 된다. 스위치를 다시 눌러 공부 모드가 종료되면, mqtt통신은 비활성화 되며, 모든 LED는 동작을 멈춘다.

시스템 개요도는 다음과 같다.



[ 시스템 개요도 ]

## 2. 구현 방법

### 2.1 하드웨어 부분

라즈베리파이에 웹 카메라를 연결한다.

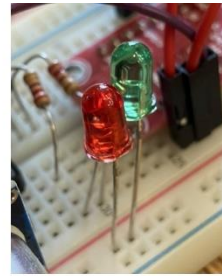
라즈베리파이에 연결된 브레드보드에 스위치, LED, 온도센서, 습도센서, 조도센서를 연결한다.



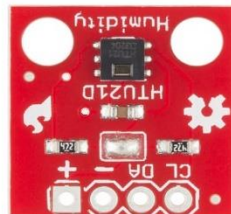
웹 카메라  
( USB 포트 )



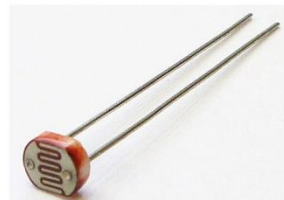
스위치  
( GPIO 21 )



LED  
( GPIO 6, 5 )

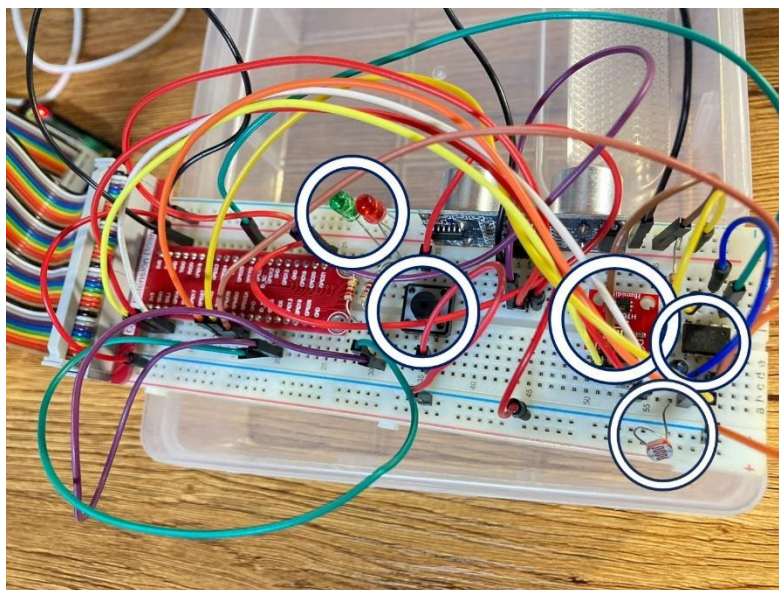


온 / 습도 센서  
( I<sup>2</sup>C )



조도 센서  
( SPI )

[ 주요 부품 ]



[ 회로도 ]

## 2.2 소프트웨어 부분

소프트웨어는 라즈베리파이 내부에서 작동하는 코드와, 로컬 서버(PC)에서 작동하는 코드로 나뉜다. 코드의 대략적 설명은 다음과 같다.

### ➤ 라즈베리파이

1. **LEDControl.py** : LED를 점등하고 소등하는 함수들을 구현하여 모듈화 한다.
2. **EnvironmentMonitor.py** : 온도, 습도, 조도 값을 측정하는 함수들을 구현하여 모듈화 한다.
3. **Camera.py** : 웹 카메라로 촬영하는 함수를 구현하여 모듈화 한다.
4. **PiController.py** : 사용자가 스위치를 누르면 공부 모드를 시작하고 mqtt 통신을 활성화 한다. EnvironmentMonitor.py로 주변 환경의 온도 / 습도 / 조도 값을 수집하여 정해진 간격으로 송신한다. 또한, Camera.py를 통해 사용자 캡처 이미지도 같이 송신한다.

공부 모드 동안에는 LEDcontrol.py를 통해 LED1을 점등하고, 온도 / 습도 / 조도 값에 대한 임계값을 설정하여, 임계값에 벗어나는 경우 LED2를 점등한다. 공부 모드가 종료되면 mqtt 통신을 비활성화 한다. 이에 따라, LED1와 LED2도 소등하게 된다.

### ➤ 로컬 서버(PC)

1. **PostureAnalysis.py** : OpenPose 모델을 통해 이미지 속 사람의 목 각도를 계산하고 관절 KeyPoint와 목 각도 값을 이미지에 표시하여 덮어씌운다. 또한, 목 각도 값을 SystemController.py에 반환한다. 각도 값으로 None값을 반환하면, 공석상태를 의미한다.
2. **SystemController.py** : mqtt 통신을 활성화 하여, 라즈베리파이로부터 이미지와 센서값들을 일정 간격으로 수신 받는다.

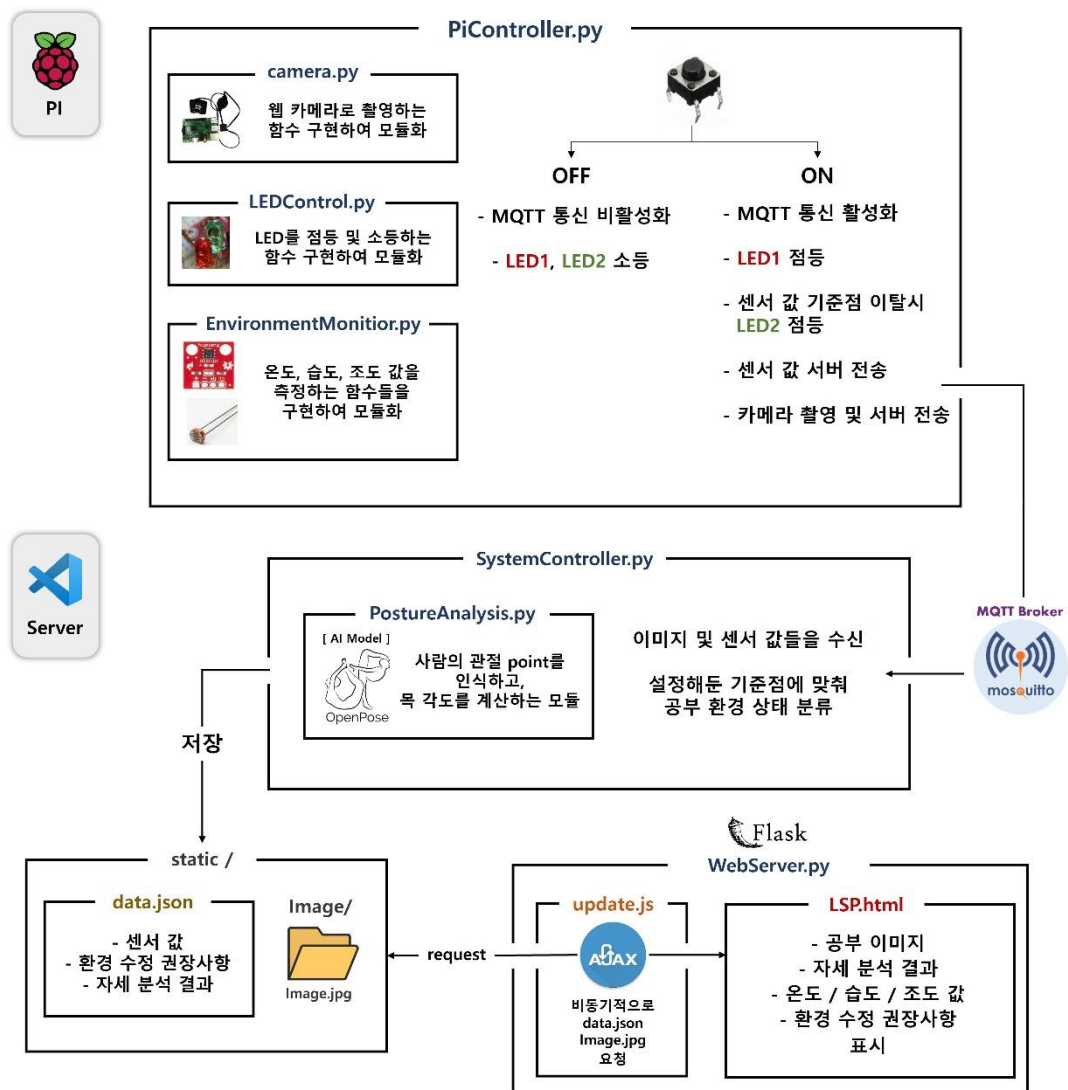
이미지는 "ori" 로컬 저장소에 저장한 후, PostureAnalysis.py를 통해 목 각도를 계산 후 "image" 디렉토리에 이미지 업데이트를 수행한다. 또한, 이 과정에서 PostureAnalysis.py 모듈로부터 반환 받은 목 각도 값이 설정 임계값에 벗어나는 경우 거북목 상태라는 정보를 data.json 파일에 저장한다.

센서 값들은 data.json 파일에 저장하고, 설정한 임계값 중 특정 구간에 해당함에 따라, 환경 수정 권장사항을 data.json에 저장한다. ( Ex : 온도센서 값 30°C 초과시, "에어컨 작동 추천!" 텍스트 저장)

3. **WebServer.py** : Flask 프레임워크를 통해 웹 서버를 구동하고, AJAX 라이브러리를 통해 비동기적으로 정보를 요청하여, LSP.html 웹페이지에 표시한다.

4. **LSP.html** : WebServer.py와 SystemController.py를 통해 정보들을 웹페이지에 표시한다.
5. **style.css** : 웹페이지의 디자인적 요소를 담고있다.
6. **update.js** : Ajax 라이브러리를 통해 비동기적으로 데이터를 요청하고, 웹페이지에 업데이트한다. 로컬 저장소 "image" 디렉토리의 이미지와 data.json파일에 저장된 센서 값들 및 수정 권장사항 텍스트 내용을 요청한다.
7. **data.json** : 센서 데이터(온도, 습도, 조도)와 환경 수정 권장사항을 저장하고, 자세 분석 결과를 저장한다. 웹 페이지에 전달하기 위한 서버의 중간 저장소 역할을 한다.
8. **Config.py** : mqtt 통신 설정값, 토픽, 저장 경로, 모델 신경망 파일, 환경 임계값을 저장한다. 중요한 데이터를 분리하여, 보안 및 유지보수를 용이하게 한다.

소프트웨어 부분의 구성도는 다음과 같다.



[ 소프트웨어 구성도 ]

### 3. 주요 소스코드

#### 3.1 PiController.py

```
import RPi.GPIO as GPIO
import paho.mqtt.client as mqtt
import EnvironmentMonitor
import camera
import LEDControl
import time
import cv2

# MQTT 설정
mqtt_client = mqtt.Client()
mqtt_broker_ip = "172.30.1.86" # MQTT 브로커의 IP 주소

# GPIO 설정
study_mode_button = 21 # GPIO21 핀에 연결된 버튼
GPIO.setmode(GPIO.BCM)
GPIO.setup(study_mode_button, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)

# 공부 모드 상태
study_mode = False

# 환경 임계값 설정
TEMP_THRESHOLD = 20 # 온도 임계값
HUM_THRESHOLD = 60 # 습도 임계값
LUMI_THRESHOLD = 400 # 조도 임계값

# 버튼 콜백 함수
def button_callback(channel):
    global study_mode
    study_mode = not study_mode
    if study_mode:
        LEDControl.control_study_mode_led(True) # 공부 모드 시작 시 빨간 LED 켜기
        print("공부 모드 시작!")
        mqtt_connect() # MQTT 통신 시작
    else:
        LEDControl.control_study_mode_led(False) # 공부 모드 종료 시 빨간 LED 끄기
        LEDControl.control_threshold_led(False) # 초록 LED도 끄기
```

```

        print("공부 모드 종료.")
        mqtt_disconnect() # MQTT 통신 종료

# 버튼 이벤트 감지 설정
GPIO.add_event_detect(study_mode_button, GPIO.RISING, callback=button_callback,
bouncetime=200)

# MQTT 연결 함수
def mqtt_connect():
    mqtt_client.connect(mqtt_broker_ip, 1883, 60)
    mqtt_client.loop_start()

# MQTT 해제 함수
def mqtt_disconnect():
    mqtt_client.loop_stop()
    mqtt_client.disconnect()

# 센서 데이터 발행 및 LED 제어 함수
def publish_sensor_data():
    if study_mode: # 공부 모드 활성화 시 실행
        temperature = EnvironmentMonitor.get_temperature()
        humidity = EnvironmentMonitor.get_humidity()
        luminance = EnvironmentMonitor.get_luminance()

        # LED 상태 결정 및 제어
        if (temperature > TEMP_THRESHOLD + 3 or temperature < TEMP_THRESHOLD - 3 or
            humidity > HUM_THRESHOLD + 30 or humidity < HUM_THRESHOLD - 30 or
            luminance > LUMI_THRESHOLD + 250 or luminance < LUMI_THRESHOLD - 250):
            LEDControl.control_threshold_led(True) # 임계값 초과 시 초록 LED 켜기
        else:
            LEDControl.control_threshold_led(False) # 임계값 내 시 초록 LED 끄기

        # MQTT를 통한 센서 데이터 발행
        mqtt_client.publish("temp", str(temperature))
        mqtt_client.publish("hum", str(humidity))
        mqtt_client.publish("lumi", str(luminance))

        # 카메라로 사진 촬영 및 MQTT를 통해 전송
        image = camera.take_picture()

```

```

        _, encoded_image = cv2.imencode('.jpg', image)
        image_bytes = bytearray(encoded_image)
        mqtt_client.publish("img", image_bytes)

# 카메라 초기화
camera.init()

# 프로그램 시작 메시지
print("Let's study Properly")
print("")

# 메인 루프
try:
    while True:
        if study_mode:
            publish_sensor_data() # 공부 모드 시 센서 데이터 발행
            time.sleep(4) # 데이터 발행 간격
except KeyboardInterrupt:
    GPIO.cleanup() # GPIO 정리
    camera.final() # 카메라 종료
    mqtt_disconnect() # MQTT 연결 해제

```

### 3.2 PostureAnalysis.py

```

import cv2
import numpy as np
import os
import math
from config import IMAGE_SAVE_PATH, PROTOFILE, WEIGHTSFILE

# OpenPose 모델을 사용하기 위한 네트워크 초기화
op_net = cv2.dnn.readNetFromCaffe(PROTOFILE, WEIGHTSFILE)

def getKeypoints(probMap, threshold=0.1):
    """ 확률 맵에서 키포인트 추출하는 함수.
    probMap: 각 키포인트 위치에 대한 확률 맵
    threshold: 키포인트를 추출하기 위한 최소 확률 임계값 """
    mapSmooth = cv2.GaussianBlur(probMap, (3, 3), 0, 0)
    mapMask = np.uint8(mapSmooth > threshold)

```



```

keypoints = []
contours, _ = cv2.findContours(mapMask, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

# 각 컨투어에 대해 가장 높은 확률을 가진 키포인트를 추출
for cnt in contours:
    blobMask = np.zeros(mapMask.shape)
    blobMask = cv2.fillConvexPoly(blobMask, cnt, 1)
    maskedProbMap = mapSmooth * blobMask
    _, maxVal, _, maxLoc = cv2.minMaxLoc(maskedProbMap)
    keypoints.append(maxLoc + (probMap[maxLoc[1], maxLoc[0]],))
return keypoints

def calculate_neck_angle(nose, neck):
    """ 코와 목 사이의 각도를 계산하는 함수.
    nose: 코의 위치
    neck: 목의 위치 """
    if nose and neck:
        horizontal_line = (1, 0) # x축 방향의 수평선
        neck_nose_line = (neck[0] - nose[0], neck[1] - nose[1])
        angle = np.arctan2(neck_nose_line[1], neck_nose_line[0]) - np.arctan2(horizontal_line[1],
horizontal_line[0])
        return np.degrees(angle)
    else:
        return None

def draw_keypoints_and_angle(frame, nose_point, neck_point, angle):
    """ 이미지에 키포인트, 선 및 각도를 그리는 함수.
    frame: 이미지 프레임
    nose_point: 코의 위치
    neck_point: 목의 위치
    angle: 계산된 각도 """
    if nose_point and neck_point:
        # 키포인트 그리기
        cv2.circle(frame, (int(nose_point[0]), int(nose_point[1])), 8, (0, 255, 255), thickness=-1,
lineType=cv2.FILLED)
        cv2.circle(frame, (int(neck_point[0]), int(neck_point[1])), 8, (0, 255, 0), thickness=-1,
lineType=cv2.FILLED)

        # 코와 목 사이의 선 그리기

```

```

        cv2.line(frame, (int(nose_point[0]), int(nose_point[1])), (int(neck_point[0]),
int(neck_point[1])), (255, 0, 0), 2)

        # 목 지점에서 수평선 그리기
        cv2.line(frame, (int(neck_point[0]) - 50, int(neck_point[1])), (int(neck_point[0]) + 50,
int(neck_point[1])), (255, 0, 0), 2)

        # 각도 표시
        cv2.putText(frame, "Angle : {:.2f}".format(angle), (int(neck_point[0] + 30), int(neck_point[1]
- 10)), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0,0,153), 2)

def process_image(image_ori_path, net=op_net, threshold=0.1):
    """ 이미지를 처리하고 목 자세 각도를 계산하는 함수.
    image_ori_path: 처리할 이미지의 경로
    net: OpenPose 모델 네트워크
    threshold: 키포인트 검출 임계값 """
    frame = cv2.imread(image_ori_path)
    frameWidth = frame.shape[1]
    frameHeight = frame.shape[0]

    # OpenPose 네트워크에 이미지 입력
    inpBlob = cv2.dnn.blobFromImage(frame, 1.0 / 255, (368, 368), (0, 0, 0), swapRB=False,
crop=False)
    net.setInput(inpBlob)
    output = net.forward()

    # 코와 목의 키포인트 검출
    nose_probMap = output[0, 0, :, :] # 코에 대한 인덱스 0
    neck_probMap = output[0, 1, :, :] # 목에 대한 인덱스 1
    nose_probMap = cv2.resize(nose_probMap, (frameWidth, frameHeight))
    neck_probMap = cv2.resize(neck_probMap, (frameWidth, frameHeight))

    nose_keypoints = getKeypoints(nose_probMap, threshold)
    neck_keypoints = getKeypoints(neck_probMap, threshold)

    # 코와 목 키포인트가 모두 있는 경우에만 계산
    if nose_keypoints and neck_keypoints:
        nose_point = nose_keypoints[0][:2]
        neck_point = neck_keypoints[0][:2]

```

```

# 각도 계산
angle = calculate_neck_angle(nose_point, neck_point)

# 90도 이상의 각도는 반대 방향으로 계산
if angle > 90:
    angle = 180 - angle

# 이미지에 키포인트와 각도 표시
draw_keypoints_and_angle(frame, nose_point, neck_point, angle)

# 이미지 저장
cv2.imwrite(IMAGE_SAVE_PATH, frame)

return angle
else:
    # 키포인트가 없는 경우 이미지만 저장
    cv2.imwrite(IMAGE_SAVE_PATH, frame)
    return None

```

### 3.3 SystemController.py

```

import paho.mqtt.client as mqtt
import PostureAnalysis # 이미지 분석 모듈 임포트
import json
from config import MQTT_BROKER_ADDRESS, MQTT_PORT, TOPICS, IMAGE_ORI_PATH,
JSON_PATH, TEMP_THRESHOLD, HUM_THRESHOLD, LUMI_THRESHOLD, ANGLE_THRESHOLD

# 데이터 저장을 위한 딕셔너리
data_storage = {
    "temp": None, # 현재 온도
    "temp_advice": "적당한 온도입니다.", # 온도에 대한 조언
    "hum": None, # 현재 습도
    "hum_advice": "적당한 습도입니다.", # 습도에 대한 조언
    "lumi": None, # 현재 조도
    "lumi_advice": "적당한 밝기입니다.", # 조도에 대한 조언
    "angle": None, # 현재 목 각도
    "angle_advice": "정상 상태", # 거북목 상태에 대한 조언
}

```

```

# JSON 파일로 데이터 저장하는 함수
def save_data_to_file():
    with open(JSON_PATH, "w") as file:
        json.dump(data_storage, file)

# 센서 데이터를 처리하는 함수들
def handle_temp(value):
    # 온도 임계값 처리
    if value > TEMP_THRESHOLD + 3:
        data_storage["temp_advice"] = "에어컨 작동 추천!"
    elif value < TEMP_THRESHOLD - 3:
        data_storage["temp_advice"] = "히터 작동 추천!"
    else:
        data_storage["temp_advice"] = "적당한 온도입니다."

def handle_hum(value):
    # 습도 임계값 처리
    if value > HUM_THRESHOLD + 30:
        data_storage["hum_advice"] = "제습기 작동 추천!"
    elif value < HUM_THRESHOLD - 30:
        data_storage["hum_advice"] = "가습기 작동 추천!"
    else:
        data_storage["hum_advice"] = "적당한 습도입니다."

def handle_lumi(value):
    # 조도 임계값 처리
    if value > LUMI_THRESHOLD + 250:
        data_storage["lumi_advice"] = "너무 밝습니다!"
    elif value < LUMI_THRESHOLD - 250:
        data_storage["lumi_advice"] = "너무 어둡습니다!"
    else:
        data_storage["lumi_advice"] = "적당한 조도입니다."

# MQTT 콜백 함수들
def on_connect(client, userdata, flags, rc):
    # MQTT 브로커에 연결됐을 때의 콜백
    if rc == 0:
        print("Broker 연결 성공\n")

```

```

# 구독 설정
for topic in TOPICS:
    client.subscribe(topic, qos=0)
else:
    print("Connection failed")

def on_message(client, userdata, msg):
    # MQTT 메시지 수신 시 콜백
    global data_storage # 전역 변수 사용

    if msg.topic == "img":
        # 이미지 데이터 처리
        with open(IMAGE_ORI_PATH, 'wb') as file:
            file.write(msg.payload)
        angle = PostureAnalysis.process_image(IMAGE_ORI_PATH)
        data_storage["angle"] = angle

        if angle is None:
            # 각도 계산 불가능한 경우 처리
            print("이미지에서 사람이 감지되지 않음")
            data_storage["angle_advice"] = "공식 상태"
        else:
            # 각도 계산 가능한 경우 처리
            print("Angle:", angle)
            if angle < ANGLE_THRESHOLD:
                data_storage["angle_advice"] = "거북목 상태입니다!"
            else:
                data_storage["angle_advice"] = "정상 상태"
    else:
        try:
            # 환경 데이터 처리
            if msg.topic in data_storage:
                value = float(msg.payload.decode('utf-8'))
                data_storage[msg.topic] = value
                print(f"{msg.topic} 값 : {value}")

            # 임계값에 따른 권장사항 생성
            if msg.topic == "temp":
                handle_temp(value)

```

```

        elif msg.topic == "hum":
            handle_hum(value)
        elif msg.topic == "lumi":
            handle_lumi(value)
    except ValueError:
        # 데이터 포맷 오류 처리
        print(f"Invalid data format for topic {msg.topic}: {msg.payload}")

# 데이터 파일로 저장
save_data_to_file()

# MQTT 클라이언트 설정
client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message
client.connect(MQTT_BROKER_ADDRESS, MQTT_PORT)

# MQTT 루프 시작
client.loop_forever()

```

### 3.4 WebServer.py

```

from flask import Flask, jsonify, render_template
import datetime
import os
import json
from config import JSON_PATH

app = Flask(__name__) # Flask 애플리케이션 인스턴스 생성

# JSON 파일에서 데이터 로드하는 함수
def load_data_from_file():
    with open(JSON_PATH, "r", encoding="utf-8") as file: # UTF-8 인코딩으로 파일 읽기
        return json.load(file)

@app.route('/')
def index():
    # 메인 페이지 라우팅
    return render_template('LSP.html') # LSP.html 렌더링

```

```

@app.route('/posture')
def get_posture():
    # '/posture' 경로 요청 시 처리하는 함수
    # 'static/image' 폴더에서 최신 이미지 파일 찾기
    image_folder = 'static/image'
    image_files = os.listdir(image_folder)
    latest_image = max(image_files, key=lambda x: os.path.getctime(os.path.join(image_folder,
x)))

    # JSON 파일에서 목 각도 관련 데이터 로드
    data = load_data_from_file()
    posture_status = data.get("angle_advice", "정보 없음") # 기본값 설정

    return jsonify({
        "image": latest_image, # 최신 이미지 파일명
        "status": posture_status # 목 각도 상태
    })

@app.route('/environment')
def get_environment():
    # '/environment' 경로 요청 시 처리하는 함수
    # JSON 파일에서 환경 데이터 로드
    data = load_data_from_file()
    return jsonify(data) # 데이터를 JSON 형태로 반환

if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0') # 서버 시작, 외부에서 접근 가능하도록 설정

```

### 3.5 update.js

```

$(document).ready(function(){
    // 주기적으로 자세 이미지와 환경 데이터 업데이트
    setInterval(function(){
        updatePosture();
        updateEnvironment();
    }, 500); // 0.5초마다 업데이트
});

```

```

function updatePosture() {
    // AJAX 요청으로 자세 데이터 가져오기
    $.get('/posture', function(data) {
        var timestamp = new Date().getTime(); // 타임스탬프 생성
        var imageUrl = 'static/image/' + data.image + '?t=' + timestamp; // 타임스탬프를 URL
        에 추가
        $('#posture-image').attr('src', imageUrl); // 이미지 소스 업데이트
        $('#posture-status').text(data.status); // 자세 상태 텍스트 업데이트
    });
}

function updateEnvironment() {
    // AJAX 요청으로 환경 데이터 가져오기
    $.get('/environment', function(data) {
        $('#temperature').text('온도: ' + Math.floor(data.temp) + '°C'); // 온도 표시
        $('#temperature-advice').text(data.temp_advice); // 온도 조언 표시
        $('#humidity').text('습도: ' + Math.floor(data.hum) + '%'); // 습도 표시
        $('#humidity-advice').text(data.hum_advice); // 습도 조언 표시
        $('#light').text('조도: ' + Math.floor(data.lumi) + 'Lux'); // 조도 표시
        $('#light-advice').text(data.lumi_advice); // 조도 조언 표시
    });
}

```

### 3.6 LSP.html

```

<!DOCTYPE html>
<html lang="ko">
<head>
    <meta charset="utf-8"/> <!-- 문자 인코딩 설정 -->
    <meta content="width=device-width, initial-scale=1.0" name="viewport"/> <!-- 반응형 웹
디자인을 위한 뷰포트 설정 -->
    <title>Let's Study Properly</title> <!-- 웹페이지 제목 -->
    <link href="/static/style.css" rel="stylesheet"/> <!-- CSS 스타일시트 연결 -->
    <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script> <!-- jQuery 라이브러리
포함 -->

    <script>
        function displayCurrentDate() {
            var today = new Date(); // 현재 날짜와 시간 생성

```



```

        var date = today.getFullYear() + '/' + (today.getMonth() + 1) + '/' + today.getDate();
// 날짜 형식 설정
        document.getElementById('current-date').innerText = date; // 날짜 표시
    }
</script>
</head>
<body onload="displayCurrentDate()"> <!-- 페이지 로드 시 날짜 표시 함수 호출 -->
    <header>
        <div class="title">
            <h1>Let's Study Properly</h1> <!-- 페이지 제목 -->
        </div>
    </header>

    <div class="container">
        <div class="study-time">
            <h2><span id="current-date"></span> 공부 환경</h2> <!-- 현재 날짜와 공부
환경 표시 -->
        </div>

        <div class="posture">
            <h2>자세 분석</h2> <!-- 자세 분석 섹션 제목 -->
            <img id="posture-image" src="" alt="자세 이미지"> <!-- 사용자의 자세를 보여주는
이미지 -->
            <p id="posture-status">상태 : Loading...</p> <!-- 자세 상태 표시 -->
        </div>

        <div class="environment">
            <div class="temperature">
                <h2>온도</h2> <!-- 온도 섹션 제목 -->
                <p id="temperature">Loading...</p> <!-- 온도 데이터 표시 -->
                <p id="temperature-advice">Loading...</p> <!-- 온도 관련 조언 표시 -->
            </div>
            <div class="humidity">
                <h2>습도</h2> <!-- 습도 섹션 제목 -->
                <p id="humidity">Loading...</p> <!-- 습도 데이터 표시 -->
                <p id="humidity-advice">Loading...</p> <!-- 습도 관련 조언 표시 -->
            </div>
            <div class="light">
                <h2>조도</h2> <!-- 조도 섹션 제목 -->

```

```

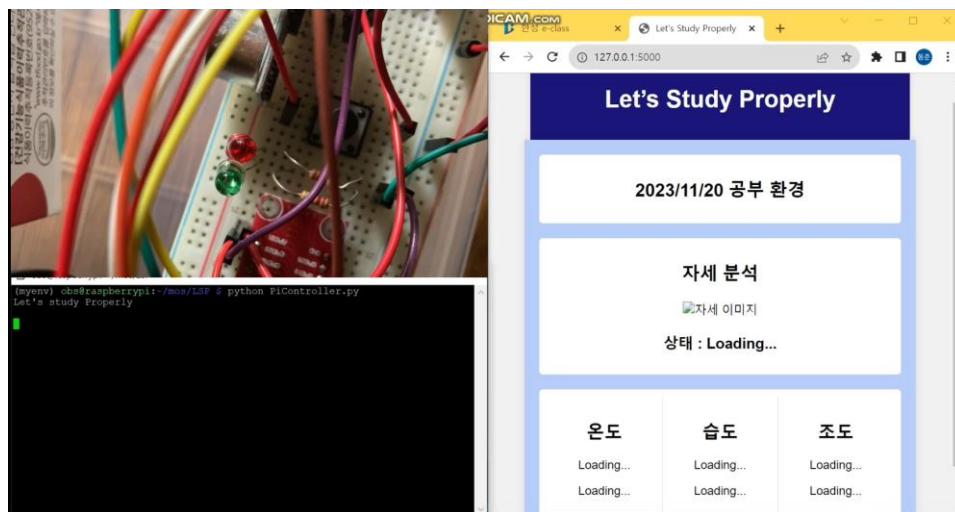
        <p id="light">Loading...</p> <!-- 조도 데이터 표시 -->
        <p id="light-advice">Loading...</p> <!-- 조도 관련 조언 표시 -->
    </div>
</div>
</div>

    <script src="/static/update.js"></script> <!-- 자세와 환경 데이터를 업데이트하는 자바스크립트 파일 연결 -->
</body>
</html>

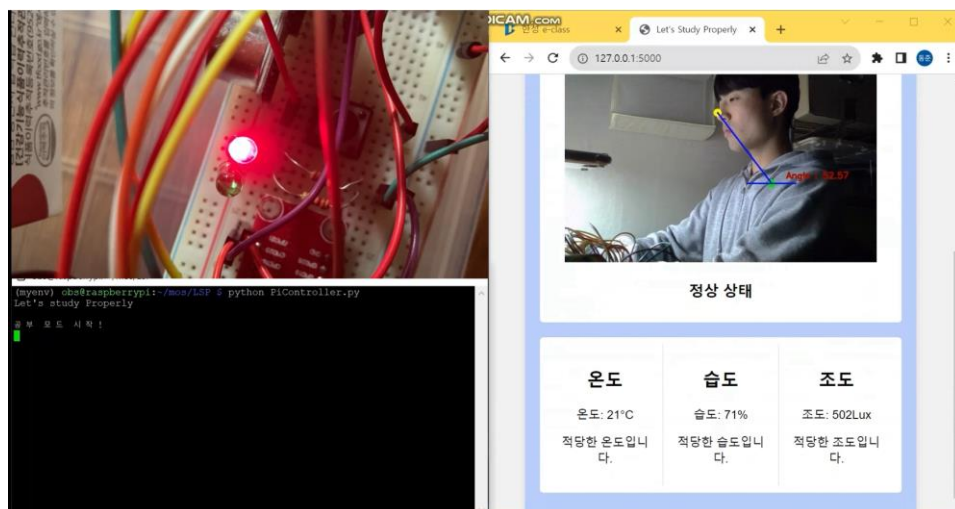
```

## 4. 구현 화면

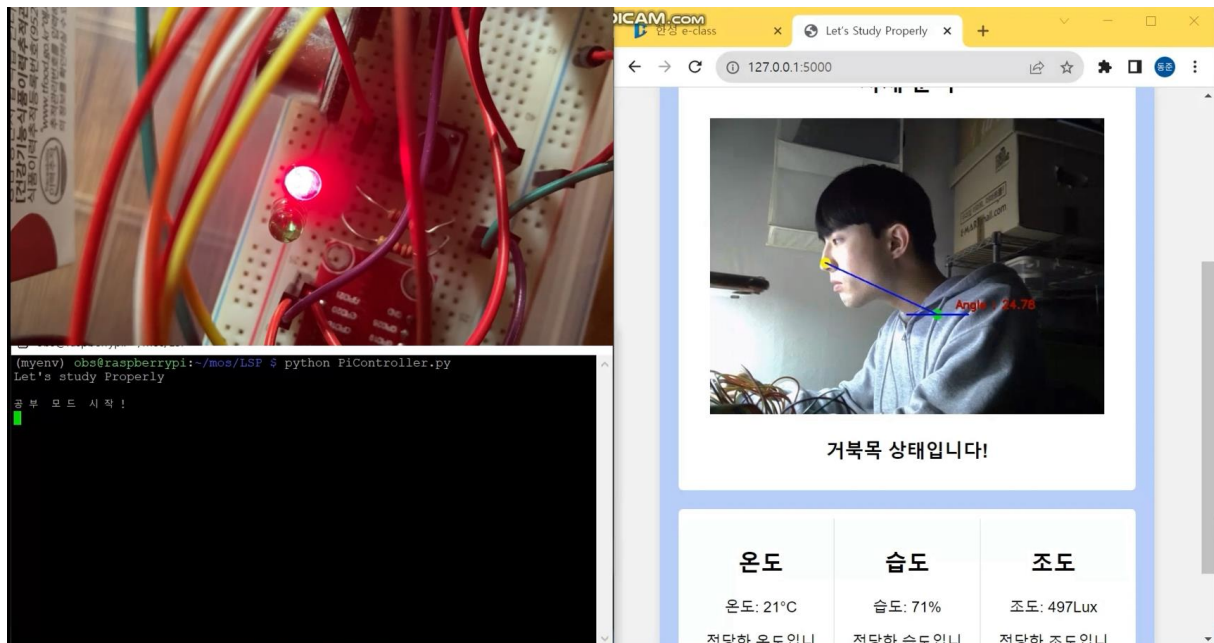
### 4.1. 초기 화면



### 4.2. 스위치를 눌러 공부모드를 작동시킨 모습



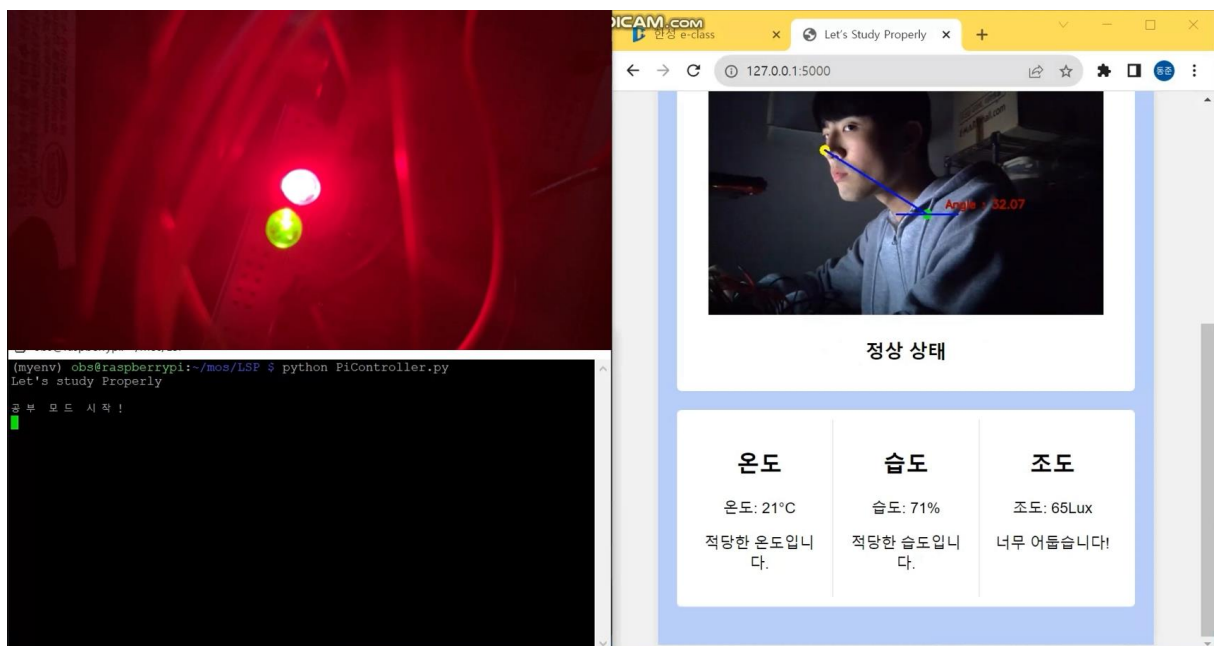
#### 4.3. 목 각도가 30도 이하로 내려가, 거북목 상태가 표시된 모습



The screenshot shows a web application interface with a camera feed of a person's head and neck. A blue line indicates the head angle, which is labeled as 24.78. Below the camera feed, the text "거북목 상태입니다!" (Turtle neck status!) is displayed. At the bottom, there are three boxes showing environmental data: 온도 (Temperature) at 21°C, 습도 (Humidity) at 71%, and 조도 (Illuminance) at 497Lux. Each box also includes a status message: "적당한 온도입니다." (Appropriate temperature), "적당한 습도입니다." (Appropriate humidity), and "적당한 조도입니다." (Appropriate illuminance).

온도	습도	조도
온도: 21°C	습도: 71%	조도: 497Lux
적당한 온도입니다.	적당한 습도입니다.	적당한 조도입니다.

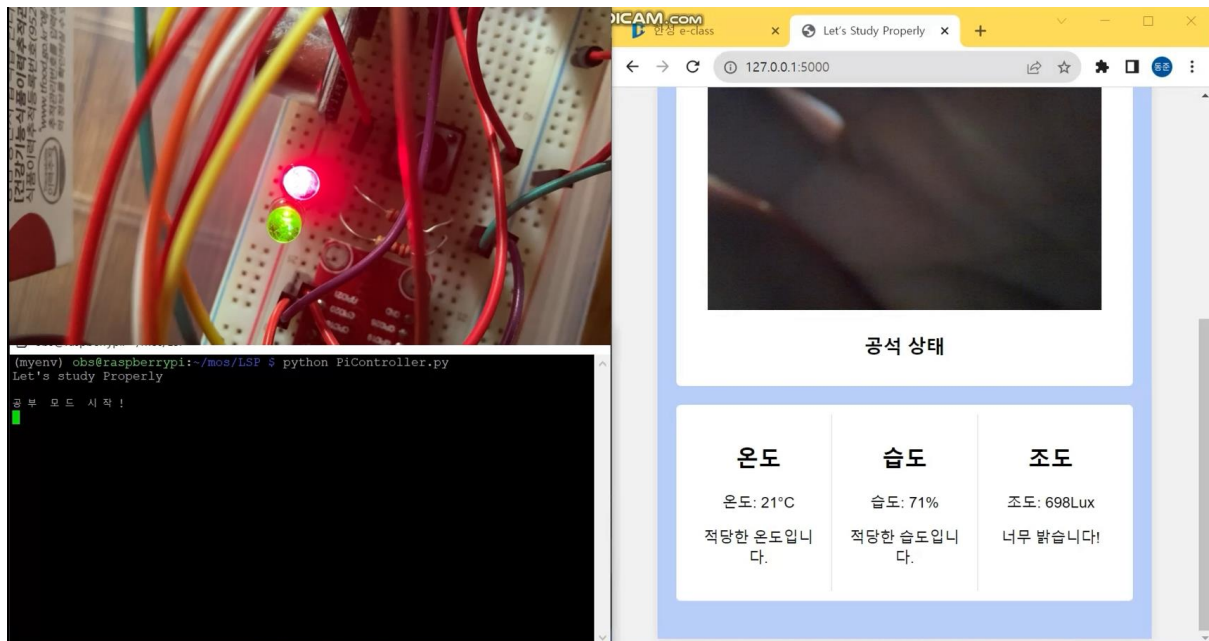
#### 4.4. 조도값이 임계점을 넘어가, 공부환경 이상상황 탐지 모습



The screenshot shows the same web application interface as in 4.3, but with a different head angle of 32.07. The text "정상 상태" (Normal status) is displayed. The environmental data boxes show the same temperature and humidity, but the illuminance has increased to 65Lux, with the status message "너무 어둡습니다!" (Too dark!).

온도	습도	조도
온도: 21°C	습도: 71%	조도: 65Lux
적당한 온도입니다.	적당한 습도입니다.	너무 어둡습니다!

#### 4.5. 사람이 탐지되지 않아, 공석 상태임을 표시한 모습



#### 4.6. 스위치를 다시 눌러, 공부모드를 종료한 모습

