**2019-2학기**

**임베디드 컴퓨팅 기말 프로젝트**

**거북사는 싫어!**



**12조**

201610052 하나리

201711309 박현정

201711343 전효진

2019.12.12 제출

1. **개요**
   1. **주제**

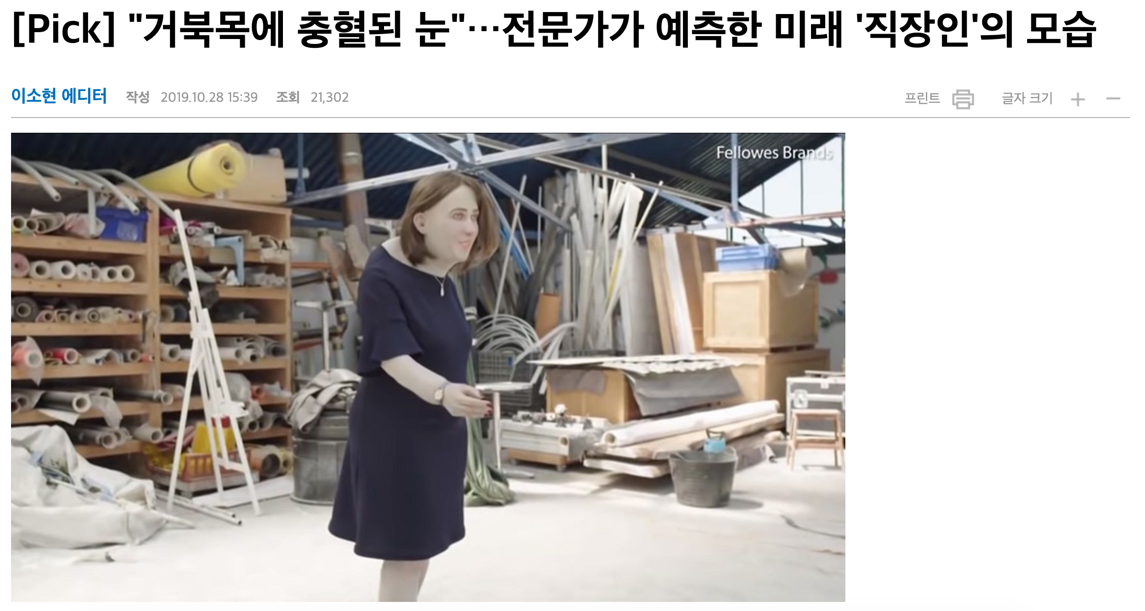


그림 <https://news.sbs.co.kr/news/endPage.do?news_id=N1005497590&plink=ORI&cooper=NAVER>

책상에 오래 앉아 공부를 하는 학생들, 업무를 보는 직장인들은 집중을 하면 자기도 모르게 목과 허리가 구부러진 채로 잘못된 자세를 취하고 있을 때가 많다. 특히 우리 소프트웨어학과 학우들은 매일 노트북 사용이 필수적인데, 눈높이가 낮은 노트북 화면을 들여다보기 위해 허리가 구부러져서 오래 앉아서 노트북을 사용하다 보면 허리 통증을 유발한다. 또 목을 쭉 빼서 모니터를 보게 되기 때문에 ‘거북목 증후군’에 대한 걱정이 많다. 거북목 증후군은 무의식적으로 머리를 앞으로 향한 채 구부정한 자세로 앉아 있어 목이 거북이처럼 앞으로 구부러지는 증상을 말한다.

보통 노트북을 사용하고 있거나 공부를 하는 친구의 자세가 좋지 않을 때, 친구에게 “허리 펴.”, “너 지금 거북목이야.” 라고 알려주곤 한다. 하지만 혼자 있을 때에는 자세가 좋지 않다고 알려줄 사람이 없어서 자세가 잘못되었다는 것을 인지하지 못한다.

‘거북사는 싫어!’ 프로젝트는 이러한 상황에서 내 목이 얼마나 앞으로 기울어져 있는 지와, 의자 등받이에서 등이 얼마나 떨어져 있는지를 센서로 측정한다. 그리고 자세가 좋지 않다고 판단되었을 때에 목과 허리가 구부러져 있는 옆모습을 실시간으로 사진을 촬영해서 LED와 함께 자동으로 알려주는 프로그램이다.

* 1. **대표 시나리오**

|  |  |
| --- | --- |
| Scenario 1 | 자세 교정 |
| Participating actors | 학생 |
| Flow of events | 학생은 프로그램을 실행한다.  기울기 센서가 연결되어 있는 안경을 착용하고, 초음파 센서가 연결되어 있는 의자에 앉는다.  공부를 시작한다.  노란색 LED와 빨간색 LED에 불이 들어온 것을 확인하면 목과 허리를 피고 자세를 고쳐 바르게 앉는다.  웹 페이지에 실시간으로 앉아있는 자세 사진이 올라오면, 오랫동안 좋지 않은 자세로 앉아있었다는 뜻이므로 자세를 고쳐 바르게 앉는다.  자세를 고쳐 바르게 앉은 후 초록색 LED에 불이 들어온 것을 확인하고 공부를 진행한다. |

|  |  |
| --- | --- |
| Scenario 2 | 스트레칭 |
| Participating actors | 회사원 |
| Flow of events | 회사원은 프로그램을 실행한다.  의자에 앉고 업무를 시작한다.  얼마나 앉아있었는지 확인하기 위해 웹페이지를 새로고침한다.  앉아있었던 시간을 확인한다.  너무 오래 앉아있었다면 피로를 풀고 거북목을 방지하기 위해 자리에서 일어나 스트레칭을 한다. |

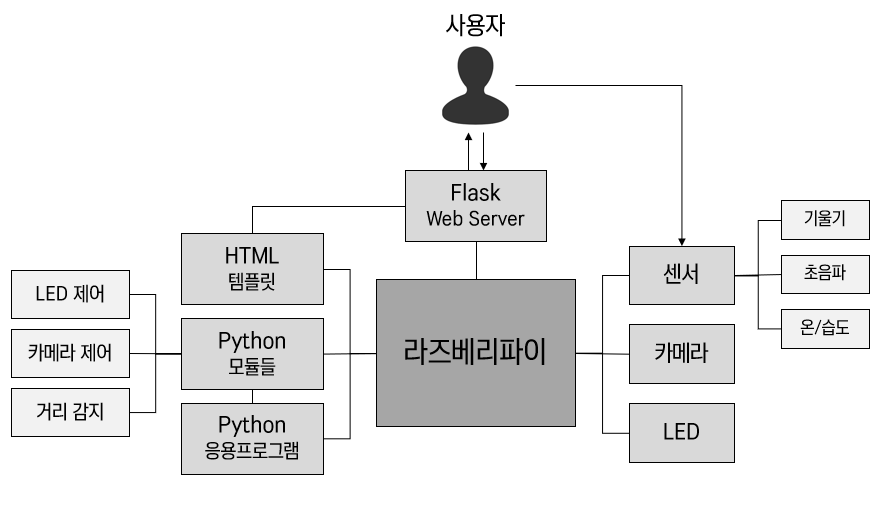
* 1. **계획서 변경 사항 및 이유**

***[계획서 변경 사항 – 주제]***

기존 1차 계획서에 작성했던 주제는 PIR 센서로 사람을 감지하고, 초음파 센서로 기계와 사람과의 거리를 측정해 사진을 찍은 후 파이썬 패키지를 이용해 찍은 사진들을 편집하는 프로젝트였다. 하지만 회의를 계속하면서 보다 다양한 센서를 이용하고, 수업시간에 배운 내용들을 충실히 적용할 수 있는 참신한 주제를 고민해보았다. 그 결과 멀티 쓰레드로 초음파 센서와 기울기 센서로 동시에 자세를 측정하는 아이디어를 생각했다. 이 프로젝트 주제가 기존 1차 계획서에 작성한 주제보다 수업 내용을 응용하기에 적합하다고 생각했고, 새로운 아이디어가 실제로 우리가 생활에서 관심을 가지고 있는 사안이므로 주제를 변경하게 되었다.

1. **시스템 설계**

**2-1. 전체 시스템 구성도**

****

**2-2. 시스템 설명**

**[하드웨어]**

1) 센서

기울기 센서, 초음파 센서, 온/습도 센서를 이용해 사용자에 대한 정보를 읽어온다

2) 카메라

라즈베리 파이에 연결한 파이카메라를 사용하여 사용자의 사진을 촬영한다.

3) LED

사용자의 자세의 심각한 정도에 따라 빨강, 노랑, 초록 불을 켜고 끄도록 라즈베리 파이 안의 LED 모듈에서 제어한다.

**[소프트웨어]**

1 )Python모듈

응용프로그램에 import할 파이썬 모듈들이다. 주로 하드웨어 센서의 제어와 측정을 위한 내용들이다. 센서에서의 값을 수치화해서 가져와, 응용프로그램에서 사용할 수 있도록 데이터화 한다.

2) Python응용프로그램

우리 시스템의 메인 응용 프로그램이다. import한 모듈들을 이용하여 센서들의 정보를 가져와 코드에서 구현한 알고리즘을 통해 현재 사용자의 자세를 분석한다. 분석한 결과에 따라 사용자의 자세를 총 5가지로 분류하여, 각각의 상태에 맞게 LED의 불을 켜도록 LED를 제어한다.

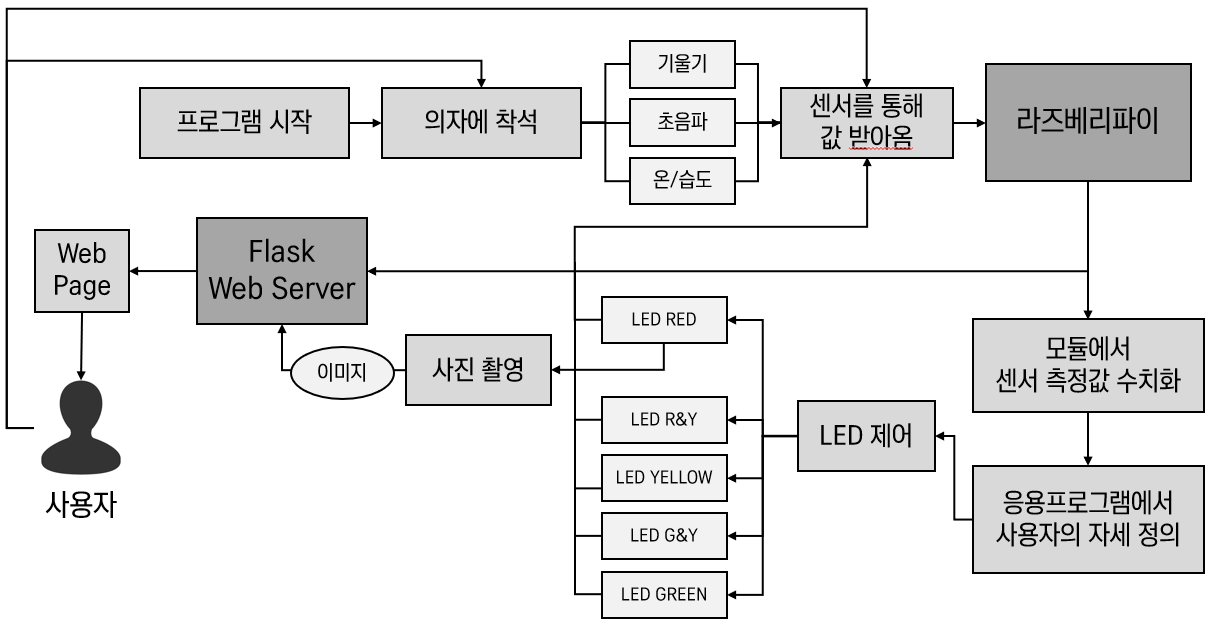
3) HTML템플릿

웹 서버에 띄울 html파일이다. 템플릿을 사용하며, 사용자가 자신의 자세 사진과 착석 시간 정보를 볼 수 있는 웹 페이지이다. html과 간단한 css코드를 사용하여 구현하였다.

**[서버]**

Flask를 사용한 웹 서버를 열고, 라즈베리 파이 안의 main.html 페이지를 서버에 띄워준다. 라즈베리 파이 안에서 이미지와 사용자의 앉은 자세에서 알 수 있는 앉은 시간을 서버로 전송한다. 클라이언트는 웹페이지에 들어가서 html 파일을 보고 정보를 얻을 수 있다.

**2-3. 대표 시나리오 작동 플로우**

****

프로그램을 시작하면 가장 먼저 사용자의 착석 여부를 알아낸다. 3가지 센서를 통해 값을 받아오고, 응용 프로그램에서 사용자의 자세를 분석한 뒤, 다섯 가지로 분류한다. 가장 안 좋은 자세일 때 LED에 빨간색 불을 켠다. 이 단계에서 사용자의 자세 정보를 갱신하고, 이에 따라 LED의 색이 계속해서 바뀐다. 만약, 빨간 불이 오래 지속된다면 안 좋은 자세로 오래 앉아 있다는 것이므로 파이카메라로 현재 사용자의 자세를 촬영한 뒤, 웹 서버에 그 이미지를 전송한다. 사용자는 서버의 웹 페이지를 통해서 자신의 자세가 어떤 상태인지 확인할 수 있다.

1. **주요 코드**

**3-1. 코드 구성도**

**스크린샷이(가) 표시된 사진

자동 생성된 설명**

**3-2. program.py**

1. AsyncTask 클래스 - Multi-Thread Programming

웹 서버 동작, 온습도 / 기울기 / 거리를 측정하는 4개의 쓰레드를 생성, 동작하게 한다.

threading.Timer(0.5,self.task1).start()를 통해 함수가 주기적으로 호출되도록 하였다.

1. hello() – 웹 서버

웹에서 루트로 접속 시 동작하는 핸들러 함수이다.

@app.route("/")

def hello():

    global starttime

    now = time.time()

    timeString = time.strftime("%H:%M:%S", time.gmtime(now-starttime))

    templateData = {'title':'I Hate Turtle!', 'time':timeString}

    return render\_template("main.html", \*\*templateData)

@app.route("/")

def hello():

    global starttime

    now = time.time()

timeString = time.strftime("%H:%M:%S", time.gmtime(now-st

now = time.time() 으로 현재 시각을 측정하고, 프로그램 시작 시 측정했던 시각과 비교해 사용자의 총 공부 시간을 계산해string으로 변환한다.

웹 페이지 상에서 총 공부 시간을 출력하기 위해 템플릿 변경 사항을 templateData에 저장, render\_template 함수로 서버에 전달한다.

1. sit() - 온습도 측정

온습도를 측정해 착석을 감지하고 총 공부시간을 계산하는 함수이다.

def sit():

    global flag2

    global instance

    result = instance.read()                #온습도 센서의 측정 결과

    temp\_pre=0

    if result.is\_valid():

        temp\_pre = result.humidity            #초기 습도

    while True:

        result = instance.read()

        if result.is\_valid():

            temp\_post = result.humidity       #현재 습도

            if temp\_post-temp\_pre>1:

                print("sitting")

                flag2=True

                break

result에 센서 측정 값을 저장한다. 초기 습도를 temp\_pre에 저장하고 습도가 증가하면 flag2를 True로 설정해 자세 측정 결과 알림을 시작한다.

1. dist() – 거리 측정

초음파 센서의 측정 오류를 예외 처리하는 함수이다.

import한 supersonic.py의 distance() 함수를 호출한다. 함수의 반환 값인 사용자 등과 등받이 사이 거리를 얻어 측정 오류 값 (2000이상)을 무시한다.

1. tilt() – 기울기 측정

안경 양쪽에 부착한 두개의 기울기 센서 값을 이용, 목의 기울기를 측정하는 함수이다.

def tilt():

    global cnt

global flag2

global dis

    t1 = GPIO.input(tilt1)                          #기울기센서1 측정 값

    t2 = GPIO.input(tilt2)                          #기울기센서2 측정 값

chk,chk2=calc(t1, t2, dis)

if flag2==True:

        if chk==23 and chk2==23:

            ledctl(chk,chk2)

            cnt += 1

            if cnt%10==0:

                print("빨간불")

                camera()

        else:

            ledctl(chk,chk2)

t1, t2에 기울기 센서 측정 값 (True/False)을 저장한다. calc 함수로 자세를 분석해 켤 led의 GPIO 핀 번호를 chk,chk2 에 저장 후, ledctl 함수로 제어한다.

빨간 불을 켤 때 (나쁜 자세) camera모듈의 camera()함수를 이용, 현재 자세를 찍는다. cnt로 사진이 너무 자주 찍히지 않게 조정한다.

1. calc() - 자세 분류

기울기 센서는 90도 이상 기울어졌을 때 기울어짐을 감지한다. 두 개의 기울기 센서의 초기 각도를 각각 45도와 75도로 다르게 설치한다. 이 결과 목의 각도를 3가지 상태로 구분해 측정할 수 있다.

초음파 센서로 등과 등받이 사이 거리를 측정해 등의 굽어짐을 판단한다. 실험 결과 거리가 20 cm 이상일 때 구부러진 자세가 되었다. 따라서 초음파 센서 측정 값이 20 이상이면 등이 굽은 자세, 20미만이면 바른 자세로 구분하기로 했다.

우리 조에서 정한 자세 분류는 아래와 같다.

|  |  |  |  |
| --- | --- | --- | --- |
|  | 45도 미만 | 45-75도 | 75도 이상 |
| 등 편 자세 | 초록 LED 점등 | 노랑 LED 점등 | 빨강 LED 점등 |
| 등 구부린 자세 | 노랑, 초록 LED 점등 | 빨강, 노랑 LED 점등 | 빨강 LED 점등 |

총 5단계로 자세를 분류하였고 거북목에 가중치를 두어 목이 75도 이상 기울어지면 무조건 빨강 LED를 점등하고 사진 촬영을 하도록 하였다. If 조건문을 이용해 자세 별 점등할 LED의 GPIO 핀번호를 반환한다.

**3-3. supersonic.py**

초음파 센서는 송신부와 수신부로 이루어져 있다. 송신부에서 초음파 펄스를 방사하고 대상물에 부딪혀 온 에코 신호를 수신부에서 받아 이 때 소요된 시간으로 거리를 산출한다.

1. 송신부 GPIO 핀을 출력모드, 수신부 GPIO 핀을 입력 모드로 설정한다.

GPIO.setup(trig, GPIO.OUT)

GPIO.setup(echo, GPIO.IN)

1. 송신부 핀을 OFF 상태로 만든다.

GPIO.output(trig, False)

1. 송신부 핀을 ON 상태로 만들면 10us 정도의 high 신호를 주면 센서가 40kHz의 펄스를 8회 발생시킨다.

GPIO.output(trig, True)

1. 송신부 핀을 OFF 상태로 만든다
2. 수신부 핀이 HIGH 상태가 되었다가 다시 LOW 상태가 되는 시간을 측정한다. 이 시간이 초음파가 방출, 반사되어 돌아와 감지되기 까지의 시간이다.

while GPIO.input(echo) == False:    #수신부 핀이 HIGH

        pulse\_start = time.time()

while GPIO.input(echo) == True:     #수신부 핀이 LOW

        pulse\_end = time.time()

pulse\_duration = pulse\_end - pulse\_start

1. <거리=속력\*시간> 이고 반사되어 돌아왔으므로 물체와의 거리는 초음파 이동거리의 절반이다.

다음 식으로 거리를 계산, 반올림 후 반환한다.

물체와의 거리 = 음속(340 m/s) 소요 시간

distance = pulse\_duration \* 17000

distance = round(distance, 2)

1. **결과물**

**4-1. 수행 환경 사진**

|  |
| --- |
| **센서, 카메라, 연결** |
| **3색 LED**  **온습도센서**  **초음파센서**  **파이카메라**  **기울기센서** |

**4-2. 주요 수행 장면 스샷**

|  |  |  |
| --- | --- | --- |
| **바른 자세(초록불)** | **조금 안 좋은 자세(노란불)** | **매우 안 좋은 자세(빨간불)** |
|  |  |  |

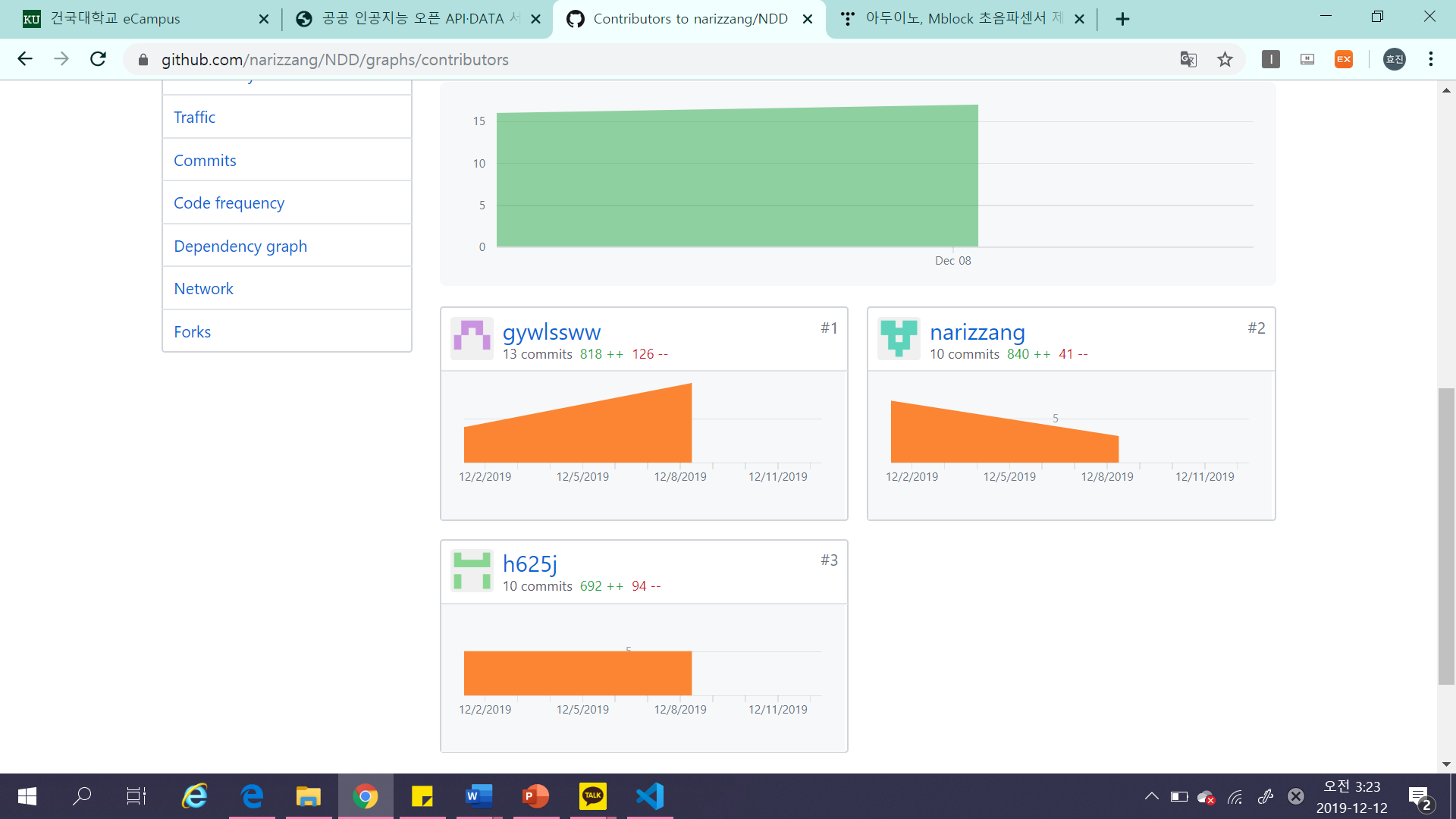
**4-3. 분석**

예상대로 목의 기울기와 등받이와 등 사이 거리에 따라 자세를 분류, 각 조건에 맞는 LED가 점등 되고 사진이 촬영되었다. 사진과 총 공부 시간은 웹 서버에 잘 출력되었다. 온습도 센서를 통해 습도가 상승한 이후 자세 분석 결과를 알려주는 부분도 잘 수행되었다.

1. **각자의 기여도**

**5-1. Github Contribution**

(박현정 h625j , 전효진 gywlssww, 하나리 narizzang)



**5-2. 조원 별 역할**

1. **박현정: Flask 웹 서버 구축, 멀티 쓰레드**
2. **전효진: 센서 측정 값 분석, 보고서 수합**
3. **하나리: GPIO 제어, 템플릿 제작, ppt**
4. **기말프로젝트를 통하여 배운 점**

6-1)기획

어떤 센서로 어떻게 자세를 측정할지가 주요 논의 사항이었다. 다양한 센서로 자세를 측정하고 싶었지만 우리가 이용할 센서들은 대부분 허용 측정 범위가 매우 작았다. 예를 들어 무게 센서, 압력 센서 등으로 착석을 감지하려 하였으나 최대 측정 무게, 압력이 너무 작아 이용이 불가했다. PIR 센서를 활용하려 하였으나 정밀한 측정이 어려웠다. 따라서 온습도 센서를 활용하는 등 대체 센서를 찾기 위해 노력해야 했다. 하드웨어를 활용한 프로그래밍을 할 때에는 다양한 부품의 상세 기능, 예제 등을 미리 조사하는 과정이 중요함을 알게 되었다. 하드웨어 제약으로 프로그래밍 방향이 좌우되었기 때문이다.

6-2)개발

1. 센서 측정

처음 기획 단계에서는 초음파 PIR 센서를 이용해 사용자 착석을 감지하려고 하였으나 정밀하게 측정하지 않았다. 착석감지를 온습도 센서를 이용하는 방법으로 변경하였다. 하드웨어를 이용한 프로그래밍은 처음이었는데, 부품 문제로 기획을 변경하게 되어 당황스러웠지만 잘 해결했다.

처음 구매한 기울기 센서는 VCC가 없어 불량인 줄 알고 다른 기울기 센서를 구매하였는데 실제로 VCC는 센서에 부착된 작은 LED를 켜는데 필요한 것일 뿐 VCC가 없는 센서도 기울기를 잘 측정하였다. 센서 내부 작동 원리를 한번 더 확인할 수 있었다.

1. 웹서버 구축

센서 측정과 웹서버를 수업시간에 배운대로 각각의 클래스로 만들어 thread를 구성하니 4개의 thread가 골고루 작동하지 않았다. 검색을 통해 모든 Thread를 하나의 클래스에 담은 뒤 Thread.timer로 클래스 안 함수를 주기적으로 호출하면 됨을 알게 되었다.

서버가 돌아가고 있는 동안에 빨간불이 들어와 사진이 찍혔지만 웹 페이지의 사진이 바뀌지 않아 헤매다가 조교님께 질문을 드렸다. 사진이 캐시에 남아있어서 바뀌지 않을 수 있다는 답변을 듣고 캐시 유효기간을 0으로 정하는 코드를 확인하고 실행해보았다. 그 결과 사진이 정상적으로 바뀜을 볼 수 있었다.

1. Multi-Thread 프로그래밍

Thread.timer를 사용해도 처음 thread를 실행시키는 순서가 중요했다. 웹 서버 thread를 먼저 실행하면 센서를 측정하는 thread들이 동작 하지 않았다. Thread 실행 순서, 함수 호출 주기를 변경해 해결하였고 운영체제 시간에 배운 스케쥴링에 대해 고민할 수 있었다.

(4) 라즈베리 파이 회로

input모드의 GPIO 포트에 대해서 풀업과 풀다운 모드에 대해서 더욱 이해할 수 있었다. 이전에 수업시간에 했던 실습에서는 버튼을 통해 제어했는데, 버튼을 누르고 뗄 때의 rising에 대해 잘 이해하지 못한 상태로 실습을 했던 느낌이 있었다. 이번 프로젝트에서는 센서에서 데이터를 받기 위해 여러개의 센서들에 데이터 선을 연결하고, 그것을 input mode GPIO에 꽂아서 풀업 모드로 설정하였다. 그렇게 하면 GPIO포트가 평소에는 계속해서 HIGH 상태였다가 데이터가 들어옴에 따라 LOW 상태로 바뀌게 되는데, 이것에 대한 구현을 하고 회로를 구성하면서 풀업 모드와 풀다운 모드에 대하여 더 자세하게 학습할 수 있었다.

6-3)총평

이번 프로젝트에서는 ‘임베디드 컴퓨팅’이라는 과목 특성에 맞게 어려운 난이도의 프로그래밍 보다는 다양한 부품들을 활용하는 데에 중점을 두었다. 수업 시간에 실습한 내용을 최대한 응용하고자 노력했으며 프로젝트를 진행하며 한 학기 수업 내용을 정리할 수 있었다. 하드웨어를 다루는 첫 수업이라 낯설기도 했지만 부품을 조립하고 코드를 구성하며 매 수업시간마다 즐겁게 임할 수 있었다. 이번 프로젝트 또한 직접 회로를 구성하고 부품들을 배치하며 재미있게 공부할 수 있었다.

1. **부록 (전체 소스코드)**

|  |
| --- |
| program.py |
| import RPi.GPIO as GPIO  import time  import threading                            #3개의 thread 생성  from flask import Flask, render\_template    #Flask 웹 서버  from camera import \*                        #사진 촬영  from tripleLED2 import \*                    #신호등 led 제어  from supersonic import \*                    #초음파 센서 측정  import dht11                                #온습도 센서 측정  cnt = 0  dis = 0  dflag=False  flag2=False  tilt1 = 20              #기울기센서1 GPIO핀을 20번으로 설정  tilt2 = 16              #기울기센서2 GPIO핀을 16번으로 설정  app= Flask(\_\_name\_\_)    #현재 모듈의 플라스크 객체 생성  @app.after\_request      #HTTP 요청 후 브라우저 응답 전 실행  def add\_header(response):      """      Add      headers to both force latest IE rendering engine or Chrome Frame      and      also to cache the rendered page for 10 minutes      """      response.headers['X-UA-Compatible']='IE=Edge,chrome=1'      #호환성 보기      response.headers['Cache-Control'] = 'public, max-age=0'     #캐시 유효시간 0      return response  @app.route("/")         #웹에서 루트로 접속 시 핸들러 동작  def hello():      global starttime      now = time.time()   #현재 시각 측정      timeString = time.strftime("%H:%M:%S", time.gmtime(now-starttime))  #전체 공부 시간      templateData = {'title':'I Hate Turtle!', 'time':timeString}      return render\_template("main.html", \*\*templateData)                 #전체 공부 시간이 출력되도록 탬플릿 변경  class AsyncTask:                                                        #멀티 쓰레드 이용을 위한 객체      def \_\_init\_\_(self):          pass        def task1(self):                                                    #thread1: Flask 서버          print("서버")          app.run(host='0.0.0.0', port=80, debug=True)          threading.Timer(0.5,self.task1).start()                         #o.5초마다 task1 호출        def task2(self):                                                    #thread2: 온습도 측정          sit()          threading.Timer(3,self.task2).start()      def task3(self):                                                    #thread3: 기울기 측정          tilt()          threading.Timer(1,self.task3).start()      def task4(self):                                                    #thread4: 거리 측정 (초음파 센서)          dist()          threading.Timer(2,self.task4).start()    def main():      at = AsyncTask()                             #각각의 task는 최초 한번은 수행해야한다        at.task2()      at.task3()      at.task4()      at.task1()                                   #웹서버를 먼저 실행하면 센서 측정이 안 되므로 마지막에 호출      def setup():                                     #GPIO핀 설정      global instance      GPIO.setmode(GPIO.BCM)      GPIO.setup(tilt1, GPIO.IN, pull\_up\_down=GPIO.PUD\_UP)      GPIO.setup(tilt2, GPIO.IN, pull\_up\_down=GPIO.PUD\_UP)      instance = dht11.DHT11(pin=21)    """      온습도 센서      -사용자 착석 감지      -착석 후 일정 시간 뒤 (습도 상승) 자세 측정 결과 led 표시  """  def sit():      global flag2      global instance      result = instance.read()                                            #온습도 센서의 측정 결과      temp\_pre=0      if result.is\_valid():          temp\_pre = result.humidity                                      #초기 습도      #print("처음 습도 : "+str(temp\_pre))        while True:          result = instance.read()          if result.is\_valid():              temp\_post = result.humidity                                 #현재 습도              if temp\_post-temp\_pre>1:                                    #습도 변화>1: 자세 측정 결과 출력                  print("sitting")                  flag2=True                  break  """      초음파 센서      -사용자 등과 등받이 거리 감지      -일정 거리 이상은 구부정한 자세      -2단계로 등의 자세 판단  """  def dist():      global dis      dis = distance()                                                    #사용자 등과 등받이 사이 거리      if (dis > 2000):                                                    #측정 오류는 무시          return  """      기울기 센서      -사용자 고개의 기울기 감지      -2개의 센서의 초기 각도를 다르게 한 후, 결과 값의 차이를 통해 3단계로 구분      -3단계로 목의 자세 판단  """  def tilt():      global cnt      global flag2                                       #습도 변화 +1 이후 자세 측정 결과 출력 위함      global dis      t1 = GPIO.input(tilt1)                             #기울기센서1 측정 값 (90도 이상: True)      t2 = GPIO.input(tilt2)                             #기울기센서2 측정 값      chk,chk2=calc(t1, t2, dis)                         #기울기, 거리 측정 값으로 자세 판단 (목, 등 구부러짐)        if flag2==True:                                    #습도 변화 +1 이후 (착석 후 일정 시간 경과)          if chk==23 and chk2==23:              ledctl(chk,chk2)              cnt += 1              if cnt%10==0:                  print("빨간불")                  camera()          else:              ledctl(chk,chk2)        #time.sleep(0.5)  """      목(3단계), 등(2단계) 자세 분류 후 5단계로 led 표시  """  def calc(t1, t2, dis):                                        #기울기 센서, 초음파 센서 측정 값      if t1 ==True and t2 == True:                               #목 75도 이상 기울어짐 (나쁜 자세)          return 23, 23                                           #빨강 LED 점등      elif t1 == True and t2==False and dis>20:                   #목 45-75도, 등 굽힌 상태 (주의3자세)          return 23, 24                                            #빨강, 노랑 LED 점등      elif t1 == True and t2==False and dis<20:     #목 45-75도, 등 핀 상태 (주의2자세)          return 24, 24                                           #노랑 LED 점등      elif t1 == False and t2==False and dis>20:                  #목 45도 미만, 등 굽힌 상태 (주의1자세)          return 24, 25                                            #노랑, 초록 LED 점등      elif t1== False and t2==False and dis<20:                   #목 45도 미만, 등 핀 상태 (바른 자세)          return 25, 25                                                  #초록 LED 점등      else:          return 0, 0                                                    #다 끄기      if \_\_name\_\_=="\_\_main\_\_":      GPIO.setwarnings(False)      setup()      ledctl(0,0)      starttime = time.time()                                             #공부 시작 시간      try:          main()                                                          #4개의 쓰레드 시작      except KeyboardInterrupt:          GPIO.cleanup() |
| camera.py |
| import time  import picamera  import web  """  파이카메라를 이용, 안 좋은 자세 지속 시 사진을 찍기 위한 파일  """  def camera():      camera = picamera.PiCamera()        #카메라 객체 생성      camera.resolution = (800,600)       #해상도 설정      camera.capture("static/"+"img.jpg") #static에 img.jpg 로 사진 파일 생성 -> 웹에서 출력      time.sleep(1)      camera.close() |
| supersonic.py |
| """  초음파 센서를 통한 거리 측정  :사용자 등과 의자 등받이 사이 거리를 측정해 자세 판단 (20cm 이상-구부정한 자세)  """  import RPi.GPIO as GPIO  import time  GPIO.setmode(GPIO.BCM)  GPIO.setwarnings(False)  trig = 6                                #초음파 센서 송신부: GPIO 6번 핀  echo = 5                                #초음파 센서 수신부: GPIO 5번 핀  GPIO.setup(trig, GPIO.OUT)              #송신부 GPIO 핀은 출력모드  GPIO.setup(echo, GPIO.IN)               #수신부 GPIO 핀은 입력모드  def distance():                         #초음파를 통한 거리 측정 함수      GPIO.output(trig, False)            #송신부 핀을 OFF 상태로 유지      time.sleep(0.5)      GPIO.output(trig, True)             #10us 정도의 high 신호 주면 초음파 센서가 40kHz 펄스 발생      time.sleep(0.00001)      GPIO.output(trig, False)            #송신부 핀 OFF 상태 전환      while GPIO.input(echo) == False:    #수신부 핀이 HIGH 되는 시점이 시작 시간          pulse\_start = time.time()      while GPIO.input(echo) == True:      #수신부 핀이 LOW 되는 시점이 수신 시간 (반시된 초음파 감지)          pulse\_end = time.time()      pulse\_duration = pulse\_end - pulse\_start    #초음파 방출-반사-감지 시간 (주기)      distance = pulse\_duration \* 17000           #거리=속력\*시간(음속:약 340m/s, 이동거리/2=물체와의 거리)      distance = round(distance, 2)               #반올림하여 소수 둘째 자리까지 나타냄        return distance |
| dht11.py |
| import time  import RPi  class DHT11Result:      'DHT11 sensor result returned by DHT11.read() method'      ERR\_NO\_ERROR = 0      ERR\_MISSING\_DATA = 1      ERR\_CRC = 2      error\_code = ERR\_NO\_ERROR      temperature = -1      humidity = -1      def \_\_init\_\_(self, error\_code, temperature, humidity):          self.error\_code = error\_code          self.temperature = temperature          self.humidity = humidity      def is\_valid(self):          return self.error\_code == DHT11Result.ERR\_NO\_ERROR  class DHT11:      'DHT11 sensor reader class for Raspberry'      \_\_pin = 0      def \_\_init\_\_(self, pin):          self.\_\_pin = pin      def read(self):          RPi.GPIO.setup(self.\_\_pin, RPi.GPIO.OUT)          # send initial high          self.\_\_send\_and\_sleep(RPi.GPIO.HIGH, 0.05)          # pull down to low          self.\_\_send\_and\_sleep(RPi.GPIO.LOW, 0.02)          # change to input using pull up          RPi.GPIO.setup(self.\_\_pin, RPi.GPIO.IN, RPi.GPIO.PUD\_UP)          # collect data into an array          data = self.\_\_collect\_input()          # parse lengths of all data pull up periods          pull\_up\_lengths = self.\_\_parse\_data\_pull\_up\_lengths(data)          # if bit count mismatch, return error (4 byte data + 1 byte checksum)          if len(pull\_up\_lengths) != 40:              return DHT11Result(DHT11Result.ERR\_MISSING\_DATA, 0, 0)          # calculate bits from lengths of the pull up periods          bits = self.\_\_calculate\_bits(pull\_up\_lengths)          # we have the bits, calculate bytes          the\_bytes = self.\_\_bits\_to\_bytes(bits)          # calculate checksum and check          checksum = self.\_\_calculate\_checksum(the\_bytes)          if the\_bytes[4] != checksum:              return DHT11Result(DHT11Result.ERR\_CRC, 0, 0)          # ok, we have valid data, return it          return DHT11Result(DHT11Result.ERR\_NO\_ERROR, the\_bytes[2], the\_bytes[0])      def \_\_send\_and\_sleep(self, output, sleep):          RPi.GPIO.output(self.\_\_pin, output)          time.sleep(sleep)      def \_\_collect\_input(self):          # collect the data while unchanged found          unchanged\_count = 0          # this is used to determine where is the end of the data          max\_unchanged\_count = 100          last = -1          data = []          while True:              current = RPi.GPIO.input(self.\_\_pin)              data.append(current)              if last != current:                  unchanged\_count = 0                  last = current              else:                  unchanged\_count += 1                  if unchanged\_count > max\_unchanged\_count:                      break          return data      def \_\_parse\_data\_pull\_up\_lengths(self, data):          STATE\_INIT\_PULL\_DOWN = 1          STATE\_INIT\_PULL\_UP = 2          STATE\_DATA\_FIRST\_PULL\_DOWN = 3          STATE\_DATA\_PULL\_UP = 4          STATE\_DATA\_PULL\_DOWN = 5          state = STATE\_INIT\_PULL\_DOWN          lengths = [] # will contain the lengths of data pull up periods          current\_length = 0 # will contain the length of the previous period          for i in range(len(data)):              current = data[i]              current\_length += 1              if state == STATE\_INIT\_PULL\_DOWN:                  if current == RPi.GPIO.LOW:                      # ok, we got the initial pull down                      state = STATE\_INIT\_PULL\_UP                      continue                  else:                      continue              if state == STATE\_INIT\_PULL\_UP:                  if current == RPi.GPIO.HIGH:                      # ok, we got the initial pull up                      state = STATE\_DATA\_FIRST\_PULL\_DOWN                      continue                  else:                      continue              if state == STATE\_DATA\_FIRST\_PULL\_DOWN:                  if current == RPi.GPIO.LOW:                      # we have the initial pull down, the next will be the data pull up                      state = STATE\_DATA\_PULL\_UP                      continue                  else:                      continue              if state == STATE\_DATA\_PULL\_UP:                  if current == RPi.GPIO.HIGH:                      # data pulled up, the length of this pull up will determine whether it is 0 or 1                      current\_length = 0                      state = STATE\_DATA\_PULL\_DOWN                      continue                  else:                      continue              if state == STATE\_DATA\_PULL\_DOWN:                  if current == RPi.GPIO.LOW:                      # pulled down, we store the length of the previous pull up period                      lengths.append(current\_length)                      state = STATE\_DATA\_PULL\_UP                      continue                  else:                      continue          return lengths      def \_\_calculate\_bits(self, pull\_up\_lengths):          # find shortest and longest period          shortest\_pull\_up = 1000          longest\_pull\_up = 0          for i in range(0, len(pull\_up\_lengths)):              length = pull\_up\_lengths[i]              if length < shortest\_pull\_up:                  shortest\_pull\_up = length              if length > longest\_pull\_up:                  longest\_pull\_up = length          # use the halfway to determine whether the period it is long or short          halfway = shortest\_pull\_up + (longest\_pull\_up - shortest\_pull\_up) / 2          bits = []          for i in range(0, len(pull\_up\_lengths)):              bit = False              if pull\_up\_lengths[i] > halfway:                  bit = True              bits.append(bit)          return bits      def \_\_bits\_to\_bytes(self, bits):          the\_bytes = []          byte = 0          for i in range(0, len(bits)):              byte = byte << 1              if (bits[i]):                  byte = byte | 1              else:                  byte = byte | 0              if ((i + 1) % 8 == 0):                  the\_bytes.append(byte)                  byte = 0          return the\_bytes      def \_\_calculate\_checksum(self, the\_bytes):          return the\_bytes[0] + the\_bytes[1] + the\_bytes[2] + the\_bytes[3] & 255 |
| tripleLED.py |
| import RPi.GPIO as GPIO  rPin=23                             #빨간 led : GPIO 23번 핀  yPin=24                             #노란 led : GPIO 24번 핀  gPin=25                             #초록 led : GPIO 25번 핀  GPIO.setmode(GPIO.BCM)  GPIO.setwarnings(False)  GPIO.setup(rPin, GPIO.OUT)           #GPIO핀 출력모드 설정  GPIO.setup(yPin, GPIO.OUT)  GPIO.setup(gPin, GPIO.OUT)    def ledctl(num1,num2):                    #색깔 별 led를 켜고 끄기 위한 함수        GPIO.output(rPin, False)      GPIO.output(yPin, False)      GPIO.output(gPin, False)      if num1!=0 and num2!=0:          GPIO.output(num1,True)          GPIO.output(num2,True) |
| main.html |
| <!--웹서버 템블릿 : 현재 자세 촬영 사진 / 전체 공부 시간 출력-->  <!DOCTYPE html>  <meta charset="utf-8">  <head>      <style>          body {              background-color: #b3c90c;          }          header {              margin-top: 20px;              margin-bottom: 30px;          }          section {              padding-top: 50px;              padding-bottom: 100px;              display:block;              background-color: #ffffff;              width : 960px;              border-radius: 90px 90px 0px 0px;          }      </style>      <center><title>{{ title }}</title></center>  </head>  <body>      <center><header>          <img src = "plogo.png" width="720">          <br>      </header></center>      <center><section>          <img src = "pimg1.png" width="720">          <img src="img.jpg" width="720">          <img src = "pimg2.png" width="720">          <br>{{ time }}      </section></center>      <footer></footer>  </body>  </html> |